NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER

Bethesda, Md. 20034

A GENERAL PURPOSE OVERLAY LOADER

FOR CDC-6000-SERIES COMPUTERS;

MODIFICATION OF THE NASTRAN

LINKAGE EDITOR

by

Roger J. Martin

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

COMPUTATION AND MATHEMATICS DEPARTMENT

RESEARCH AND DEVELOPMENT REPORT

April 1973

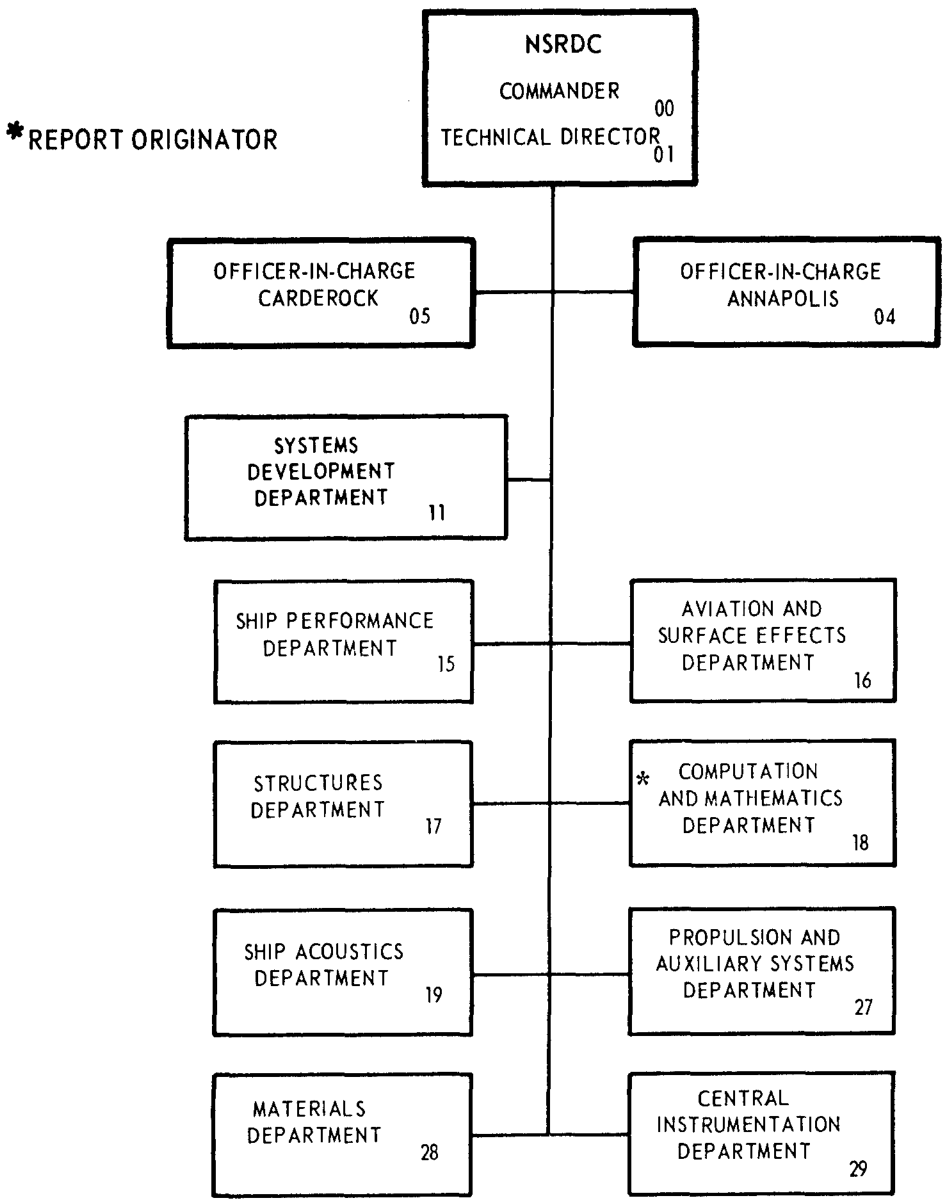
Report 4062

A GENERAL PURPOSE OVERLAY LOADER FOR CDC 6000-SERIES COMPUTERS; MODIFICATION OF THE NASTRAN LINKAGE EDITOR

The Naval Ship Research and Development Center is a U. S. Navy center for laboratory effort directed at achieving improved sea and air vehicles. It was formed in March 1967 by merging the David Taylor Model Basin at Carderock, Maryland with the Marine Engineering Laboratory at Annapolis, Maryland.

Naval Ship Research and Development Center
Bethesda, Md. 20034

MAJOR NSRDC ORGANIZATIONAL COMPONENTS



NDW-NSRDC 3960/43b (Rev. 3-72)
GPO 928-108

DEPARTMENT OF THE NAVY
NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER
BETHESDA, MD. 20034

A GENERAL PURPOSE OVERLAY LOADER FOR CDC 6000-SERIES COMPUTERS; MODIFICATION OF THE NASTRAN LINKAGE EDITOR

by

Roger J. Martin

April 1973 Report 4062

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The NASA Structural Analysis (NASTRAN) Linkage Editor is a general purpose linkage editor designed to execute on CDC 6000-series computers. It provides a means of utilizing available main memory to accommodate large programs which normally will not fit into the available main memory. As originally designed, the NASTRAN Linkage Editor required RUN FORTRAN compiled input. This report describes a modified and improved version of the Linkage Editor which has been extended to accept either RUN FORTRAN compiled or FORTRAN EXTENDED compiled input.

ADMINISTRATIVE INFORMATION

The work reported here was performed within the Computer Sciences Division of the Computation and Mathematics Department. It was carried out under Task Area ZF0990101, Work Unit 1-1844-007 sponsored by the Office of the Director of Navy Laboratories (DNL) through the Navy NASTRAN Systems Office, Code 1844, Naval Ship Research and Development Center.

BACKGROUND

The NASTRAN Linkage Editor was designed to provide an efficient load capability for NASTRAN jobs being run on the CDC 6000 series computers. It was limited, however, to input compiled using the RUN FORTRAN compiler. Since RUN is being phased out, and since it was desired to use input compiled using the FORTRAN EXTENDED (FTN) compiler, this project was initiated to modify the Linkage Editor to accept FTN input. In addition, the Linkage Editor was to be converted to FTN compilable code.

A detailed description of the modifications made to the Linkage Editor is given in A-pendix A. Excerpts from the NASTRAN Programmer's Manual[1] have been included in Appendixes B and C for the user's convenience. Further details concerning the design of the Linkage Editor may be found in the Manual, pp. 7.1-1 through 7.2-206.

The Linkage Editor and the system routines needed for LINKLIB are maintained on both the NSRDC CDC 6700 and the CDC 6400 computer systems. For further information, contact: User Services Office
Code 1892.1
Naval Ship Research and Development Center
Bethesda, Maryland 20034

1 "The NASTRAN Programmer's Manual," Edited by F. J. Douglas, National Aeronautics and Space Administration Report NASA SP223, Washington, D.C. (Sep 1970).

INTRODUCTION

The NASTRAN Linkage Editor is a general purpose linkage editor designed to utilize memory storage efficiently for medium to large programs. By using this Linkage Editor, a job which can be logically structured into segments can be run using less memory, since the entire job does not have to be present in the user's field length at any one time.

The Linkage Editor allows the user to divide a program into subprograms which can be assembled or compiled independently. These subprograms can then be combined into a link with contiguous storage addresses. The link is written onto a random access file for immediate access. Since the Linkage Editor can process more than one link per job, each link is written with a unique link number.

During creation of the link, the relocatable binary code of the user program is inserted into the link as directed by the control cards. A library search is conducted for external references not contained on the user library file.

In order to minimize main storage requirements, a programmer can arrange a program into an overlay structure divided into segments. Two or more segments which need not be in core simultaneously can be assigned the same storage addresses in different links and can then be loaded at different times.

The Linkage Editor can produce a storage map and a cross-reference table of the subprograms in each link.

PROJECT DESCRIPTION

A flow chart showing input to and output from the Linkage Editor is given in Figure 1 on the opposite page.

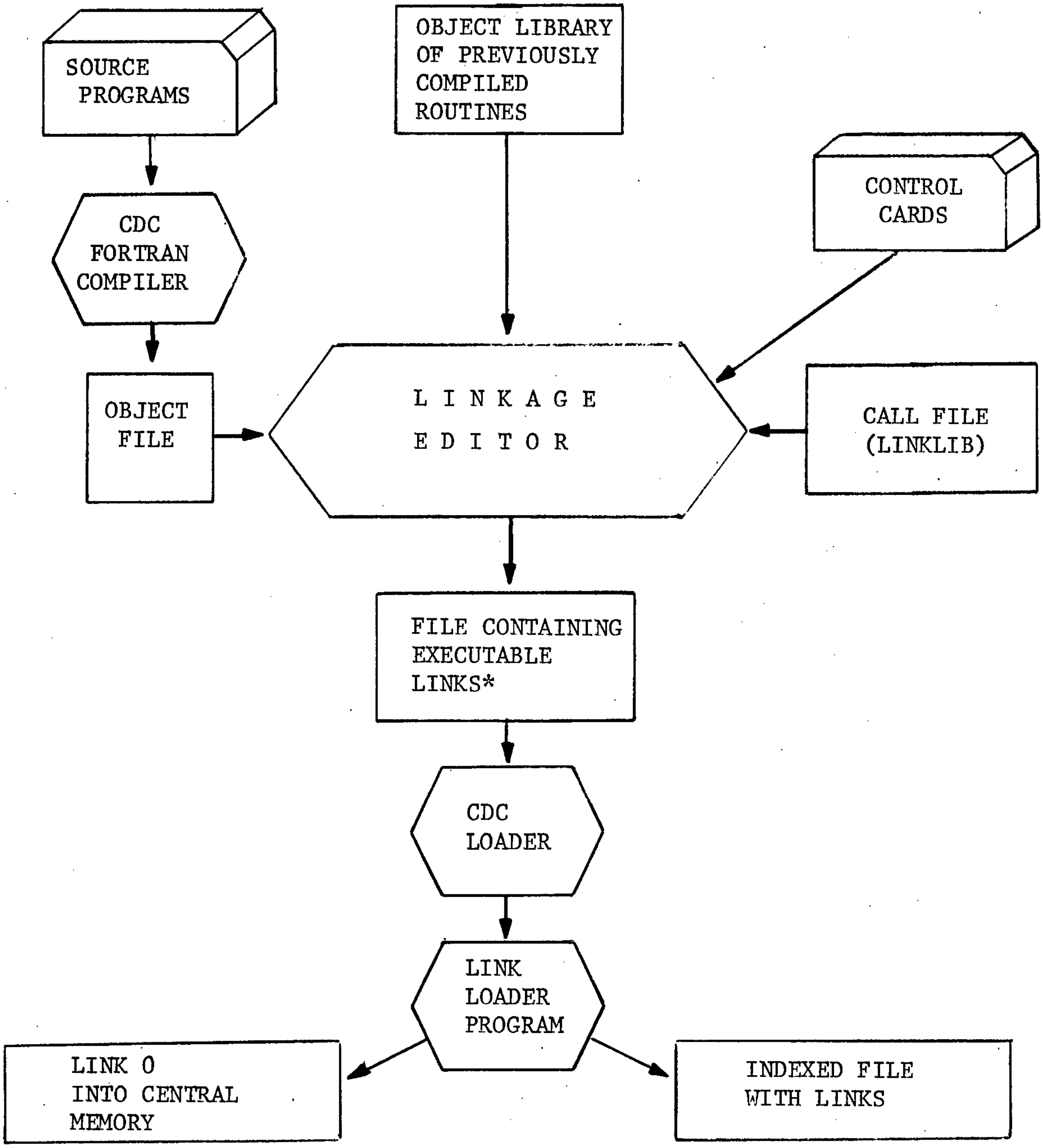
The work of this project was logically divided into two phases.

Phase 1 - Convert the Linkage Editor to accept FTN compiled jobs.

Phase 2 - Convert the Linkage Editor code to FTN.

The original version of the Linkage Editor would not link-edit FTN compiled jobs. Many minor problems existed, but the major fault was with the handling of replication (REPL) tables in the relocatable binary which was being link-edited. This problem required extensive research into the structure of relocatable binary tables and was solved by making changes to a number of routines. The main changes were made to REPLTAB which performs the actual expansion of replication tables. A description of the modifications made to the Linkage Editor may be found in Appendix A.

The majority of the changes made during Phase 2 were to the COMPASS language routines. One major problem involved the transfer of arguments from FORTRAN to COMPASS subroutines. In RUN, the addresses of the arguments are passed in the B registers while in FTN, Register A1 points to a list of argument addresses. When only one or two arguments were involved, or the routine was very short, the code was changed to properly pick up the arguments. When several arguments were passed, the B registers were set with the addresses of the arguments and no further modification was made to the code.



* The Linkage Editor will either update an existing file with new LINKS or create a complete new file.

Figure 1 - Linkage Editor Input and Output

A second major problem was the preservation of Register A0. With RUN, since there was no need to save Register A0, A0 was often used as a scratch register. To avoid such use, the code was changed to omit use of Register A0. Otherwise, Register A0 is saved on entry to a routine and restored just prior to returning to the calling program.

In addition to the conversion of the Linkage Editor to FORTRAN EXTENDED, a number of other enhancements have been made and many minor bugs found and eliminated.

(1) Dynamic allocation of memory. When it is not necessary to maintain the maximum field length, memory can be dynamically adjusted to the size needed for the current link. This feature is disabled by default but can be enabled by including the following control card in the Linkage Editor control cards for LINK 0.

RENAME LINK=LINK$

Note: The NOREDUCE option must still be used to link edit the program. It need not be used to execute the user's link edited program (OUTFILE) if memory is to be dynamically allocated. If LINK is not renamed LINK$, the NOREDUCE option must be used.

(2) New end-of-card delimiter. The "end-of-card" control character has been changed from "$" to "*". This was done to allow routines to be renamed with FTN routine names which end in "$".

(3) New OUTFILE codes. The following codes may be used to indicate what form of linkage editor output is to be created:

S = T = sequential file

R = C = random file

(4) A link-edited version of the Linkage Editor has been produced which requires even fewer words of memory, although the execution time is slightly longer. Assuming the use of the default values for the parameters, the Linkage Editor requires 64000_8 words of memory, while the link-edited version requires only 57000_8.

FEATURES AND FUNCTIONS

The NSRDC version of this linkage editor has the following features:

- An unlimited number of overlay levels.
- Implicit segment loading. The user can describe the overlay structure to the linkage editor through control cards. This allows the program to be structured after it has been coded.
- Complete communication is maintained between all levels of overlay.
- Named common blocks can be explicitly positioned.
- All segments are maintained on a random file. This provides immediate access to a needed segment.
- Either FTN or RUN-compiled input may be used as input to the linkage editor.
- Individual links of a LINKEDIT OUTFILE may be updated without relinking the entire program.
- Dynamic allocation of memory as each link is loaded is available.

The linkage editor has five separate functions:

1. Combine assembled or compiled subprograms into links suitable for loading and execution.
2. Resolve undefined externals using a library file.
3. Rearrange control sections (subprograms) and rename external references through the use of control statements.
4. Reserve common block space for each common area generated by FORTRAN or COMPASS.
5. Provide processing options and diagnostic messages.

USING THE LINKAGE EDITOR

There are two prerequisites to the use of the Linkage Editor:

- The program to be link-edited must have a structure which can be divided into independent or semi-independent segments.
- There must be a library (LINKLIB) which contains the system routines and other routines which are to be used to resolve unsatisfied external references.

If these prerequisites are met, take the following steps:

<u>Step 1</u>. Structure the source program into segments in the form of a tree structure.

<u>Step 2</u>. Define the tree structure of the program with Linkage Editor control cards.

<u>Step 3</u>. Create user libraries of compiled routines.

<u>Step 4</u>. Create a call file (LINKLIB) which contains the needed system routines (LINKLIB supplied with the inkage ditor) and user routines which are to be used to resolve external references.

<u>Step 5</u>. Execute the Linkage Editor to create the link edited OUTFILE.

LINKAGE EDITOR CONTROL COMMANDS

The commands discussed in this section are the only commands which will be accepted by the Linkage Editor. Note the following:

- The LINKEDIT command must always be first and it must be followed by a LIBRARY command.

- Definition of a link is begun with the LINK command and ended with END. The last command must always be ENDLINKS.

- Comments may be inserted after a command by using an asterisk ("*") as an end-of-card delineator. Example:

```
LIBRARY LGO  * THIS IS A COMMENT
```

Several terms which are used generally throughout the individual descriptions are explained here.

Control Section	A control section consists of all the instructions and data defined for a subprogram or common block.
Segment	A segment is the smallest functional unit (one or more control sections) which can be loaded as a logical entry during execution.
Region	A region is a contiguous area of main memory reserved for specific segments.
Link	A link is a set of one or more segments which comprise a logical subdivision of the program.

LINKEDIT

```
LINKEDIT  INFILE = name (a), OUTFILE = name (b),
          LET, NOLIST, NOMAP, XREF,
          PARAM (i) = n
```

Command Description:

The LINKEDIT command specifies input and output file names and status, what processing is to be performed, and sizes of parameters.

Parameters:

INFILE	- Previously produced Linkage Editor file which is to be updated during this run.
OUTFILE	- File on which executable link edited file is to be written.
a, b	- R or C indicates the file is a random file or disk. S or T indicates the file is a sequential file.
LET	- Directs the Linkage Editor to ignore non-fatal errors.
NOLIST	- Suppresses listing of control statements.
NOMAP	- Suppresses storage maps.
XREF	- Generates external reference tables as specified by PARAM (7).
PARAM (1)	- Length of FET + buffer for all files (Default: 530).
PARAM (2)	- Maximum number of object decks in all libraries (Default: 1000).
PARAM (3)	- Maximum size of any table in object deck (Default: 500).
PARAM (4)	- Maximum number of links (Default: 32).
PARAM (5)	- Maximum number of segments per link (Default: 128).
PARAM (6)	- Maximum length of a control section for which text is defined (Default: 5000).
PARAM (7)	- XREF Options (Default: 3). = 1: References from each subprogram = 2: References to each subprogram = 3: Both 1 and 2
PARAM (8)	- Intermediate table printout option (Default: 0) = 0: Don't print tables = 1: Print tables

Notes:

- The LINKEDIT command must be the first input command. Only one LINKEDIT command is allowed per job step.
- If XREF is selected, the status of INFILE and OUTFILE must be S or T.
- NOMAP is ignored when XREF is selected.
- If INFILE = OUTFILE, the status of the files must be the same.
- If the status of INFILE is R, a new Scope file is not created. Therefore, the permanent file must be EXTENDed if the updates are to be made permanent. Remember that the old copy of INFILE will not exist after an update run of this type.

Examples:

```
LINKEDIT  OUTFILE = TAPE(S), LET, XREF, PARAM (7) = 3

LINKEDIT  INFILE = OLDLKED(S), OUTFILE = NEWLKED(S)

LINKEDIT  OUTFILE = ROGER(R), PARAM (6) = 8000
```

- -

LIBRARY

LIBRARY libname1 = name1, name 2, ... /libname2/libname3 = name3

Command Description:

The LIBRARY command names all files which may be used on INCLUDE commands. It must always be in the second input command. There may be only one LIBRARY command per job step.

Parameters:

libname1 = name1, name2, ... : Files name1 and name2 may be concatenated and renamed libname1. If duplicate deck names occur on the files, the first one found will be used.

libname2 Files may be referred to by their actual local name ... i.e., libname.

libname3 = name3: File name 3 may be renamed libname3.

Notes:

(1) The file names are not actually changed, but are renamed only for Linkage Editor reference.

(2) "/" is used as a delimiter.

Examples:

LIBRARY LGO = LGO, OLDLIB

All references to LGO in INCLUDE commands will cause both LGO and OLDLIB to be searched.

LIBRARY LGO

LGO is the only file name which may appear on an INCLUDE command.

LIBRARY LINKED = LGO

All references to LINKED will cause LGO to be searched.

LIBRARY LGO = LGO, OLDLIB/MYFILE/LINKED=OLD

- -

LINK

LINK n

Command Description:

The LINK command directs the Linkage Editor to begin processing link n. The first LINK command must immediately follow the LIBRARY command. Additional LINK commands may follow the END command of the current link description. Whenever LINK 0 is processed, it must be processed first.

Parameter:

n - A non-negative integer link number.

Example:

LINK 0

INCLUDE

INCLUDE libname (deck, BLKDATA(comname))

Command Description:

The INCLUDE command directs the Linkage Editor to include all the named object checks from the specified library in the current link. This command may appear anywhere between the LINK and END commands for a link. Subprograms are included in the order found.

Parameters:

libname - Specifies the name of a sequential file listed in the LIBRARY command.

deck - Specifies the name of an object file which is to be included in this link from libname.

BLKDATA - Indicates named common areas are to be included.

comname - Specifies the name of the first mentioned named common block in the BLOCK DATA subroutine.

Note:

While FTN allows BLOCK DATA subroutines to be given any name, the Linkage Editor requires the BLOCK DATA subroutine be either unnamed or have the first six (6) characters of the name be "BLKDAT".

Examples:

```
INCLUDE   LGO (SUB1)

INCLUDE   LGO (SUB2, SUB3, SUB4)

INCLUDE   LINKED (BLKDATA(COM1))

INCLUDE   MYFILE (SUB5, SUB6, BLKDATA(COM2))
```

- -

REGION

REGION

Command Description:

The REGION command defines the start of a new region. It may be used anywhere within a link definition except in LINK 0.

- -

OVERLAY

OVERLAY name

Command Description:

The OVERLAY command indicates the beginning of an overlay segment. It may appear anywhere in a link description, but may not be used in LINK 0.

Parameter:

name - Specifies a symbolic name which indicates the origin of a segment. It is not related to external symbols in the link.

Notes:

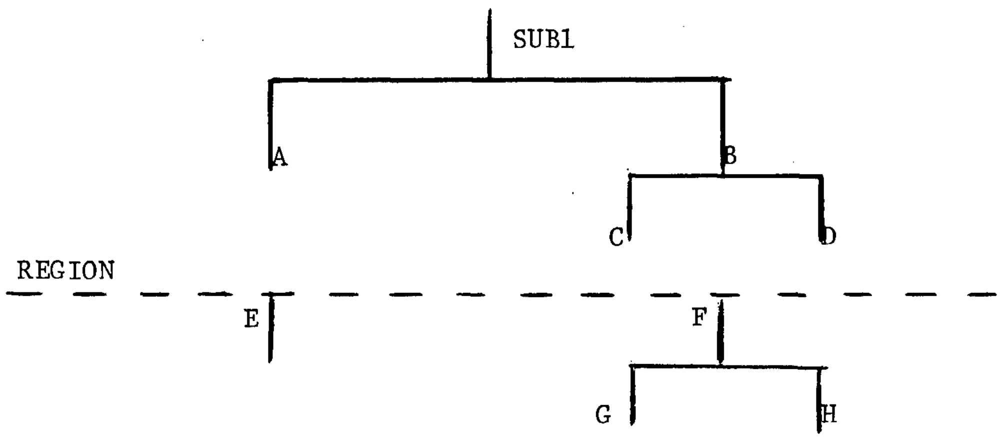
(1) An overlay should be specified when two or more routines which do not have to be in memory simultaneously are needed.

(2) Common blocks should be positioned at the end of the longest overlay.

(3) Overlay names are defined for each region. Therefore the same level overlay may have different names in different regions of the same link.

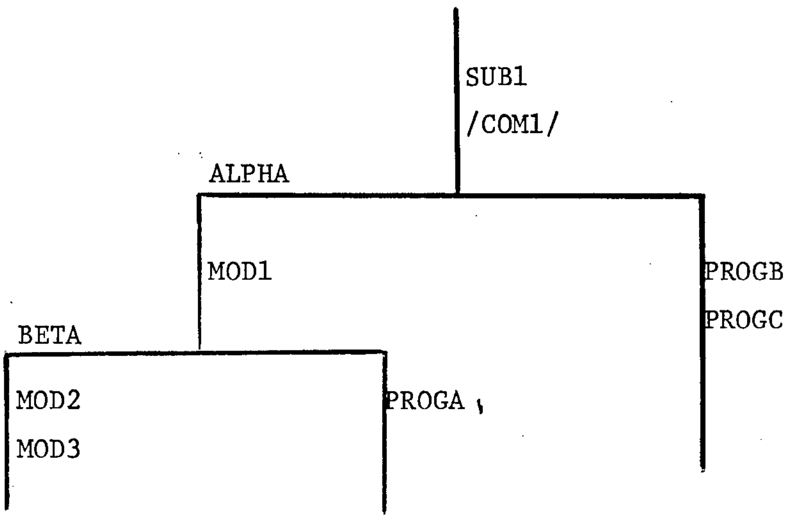
Examples:

Example 1 - Multiple Region Overlay




```
REGION
OVERLAY ALPHA1
INCLUDE LGO (A)
OVERLAY ALPHA1
INCLUDE LGO (B)
OVERLAY BETA1
INCLUDE LGO (C)
OVERLAY BETA1
INCLUDE LGO (D)
REGION
OVERLAY ALPHA2
INCLUDE LGO (E)
OVERLAY ALPHA2
INCLUDE LGO (F)
OVERLAY BETA2
INCLUDE LGO (G)
OVERLAY BETA2
INCLUDE LGO (H)
```

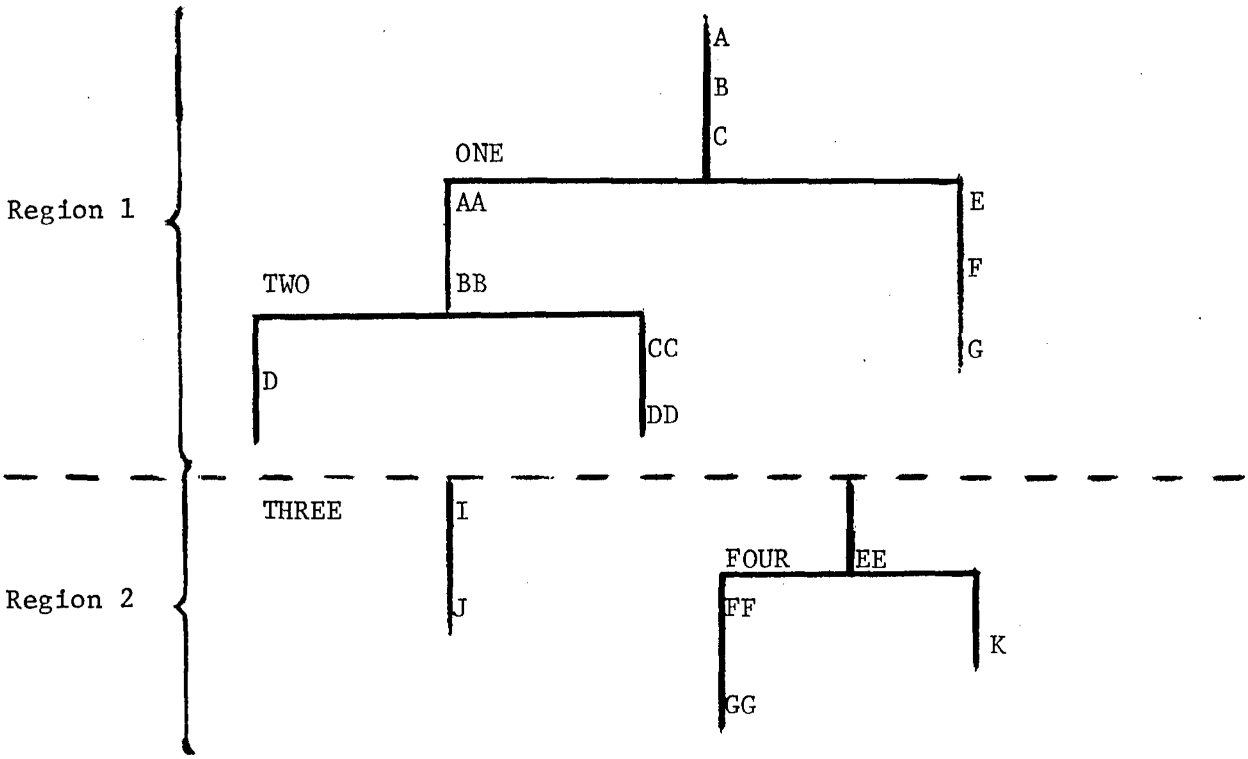
Example 2* - Single Region Structure (No REGION command needed)




```
INCLUDE   MASTER(SUB1)
INCLUDE   MASTER(BLKDATA(COM1))
OVERLAY   ALPHA
INCLUDE   NEWDCKS(MOD1)
OVERLAY   BETA
INCLUDE   NEWDCKS(MOD2, MOD3)
OVERLAY   BETA
INCLUDE   MASTER(PROGA)
OVERLAY   ALPHA
INCLUDE   MASTER(PROGB, PROGC)
```

* Examples 2 and 3 have been taken from the NASTRAN Programmer's Manual

Example 3 - Multiple Region Structure




```
INCLUDE  OBJ(A, B, C,)
OVERLAY  ONE
INCLUDE  DECKS(AA, BB)
OVERLAY  TWO
INCLUDE  OBJ(D)
OVERLAY  TWO
INCLUDE  DECKS(CC, DD)
OVERLAY  ONE
INCLUDE  OBJ(E, F, G)
REGION
OVERLAY  THREE
INCLUDE  OBJ(I, J)
OVERLAY  THREE
INCLUDE  DECKS(EE)
OVERLAY  FOUR
INCLUDE  DECKS(FF, GG)
OVERLAY  FOUR
INCLUDE  OBJ(K)
```

INSERT

`INSERT name`

Command Description:

The INSERT command positions control sections within an overlay segment. It is used following an OVERLAY command that defines the segment in which the control section is to be placed.

Parameter:

name - Specifies the name of the control section to be inserted.

Note:

(1) If the control section is inserted more than once within a link, the last INSERT will be honored and all others ignored.

Examples:

```
INSERT   SUB1

INSERT   SUB2, SUB3
```

- -

RENAME

```
RENAME   oldname = newname
RENAME   oldname (subprogram) = newname
```

Command Description:

RENAME changes external references to a name either throughout a program or within a subprogram. It may appear anywhere within a link description.

Parameters:

old name - Symbol which is externally referenced.

new name - Symbol to which the reference is to now be made.

subprogram - Name of the subprogram in which the rename is to be performed.

Notes:

(1) RENAME does not actually change the symbol name, but switches external references to the new name.

(2) Only one rename may be specified on a single command.

Examples:

```
RENAME  SORT = SORTXX

RENAME  LINK = LINK$
```

ENTRY

ENTRY name

Command Description:

ENTRY defines which control section will be branched to when a link has been called.

Parameter:

name - Control section name.

Notes:

- The control section must be in the root segment of the links.
- In Link 0, the entry name must be the main program.
- Each link must have one and only one ENTRY command.

Examples:

ENTRY MAIN

ENTRY SUB1

END

END

Command Description :

END defines the end of a set of control statements for a link. It must be placed immediately after the last control command for a link description.

ENDLINKS

ENDLINKS

Command Description :

ENDLINKS defines the end of the link editor control statements. This command is the last one on the input file. It should be preceeded by an END command for the last link definition.

LINKLIB - THE CALL LIBRARY

LINKLIB is the call library used by the Linkage Editor to resolve external references which cannot be resolved from the routines listed on INCLUDE commands. A LINKLIB must always be used when the Linkage Editor is executed.

The LINKLIB supplied with the Linkage Editor contains all the necessary system routines for both FTN and RUN compiled routines. If user routines are to be used to resolve external references, the user routines should be confirmed with the supplied call library on a file named LINKLIB. <u>The call library must be called LINKLIB</u>.

EXECUTION OF THE OUTFILE

The output (OUTFILE) of the Linkage Editor is produced in one of two forms:

- As a sequential binary file (status = T or S)
- As an indexed random file (status = R or C)

One of the following three Scope control card formats should be used to execute the outfile.

1. OUT1.CATLOG(OUT2)

This form is used when OUTFILE = OUT1(S) was used on the LINKEDIT card and an indexed random form of the file is wanted.

The new random file (OUT2) is not executed.

2. OUT2.ATTACH

This form executes the indexed random file created with OUTFILE = OUT2(R) in the LINKEDIT card or with OUT1. CATLOG(OUT2) described in (1) above.

3. OUT4.

This form is used when OUTFILE = OUT4(T) is specified on the LINKEDIT file. This control card causes the bootstrap routine to generate the following control cards:

OUT4.CATLOG(SYSLMOD)

SYSLMOD.ATTACH

The OUTFILE is changed to an indexed random form and then executed.

LINK-EDITED VERSION OF THE LINKAGE EDITOR

The Linkage Editor has itself been link-edited. This has resulted in a field length reduction of approximately 5100_8 words.

A diagram of the link-edited structure is shown in Figure 2. The Scope control cards used and the output received are reproduced in the pages following the figure.

This output is provided merely as one example of a specific application of the Linkage Editor. Other general examples are provided in Appendix C.

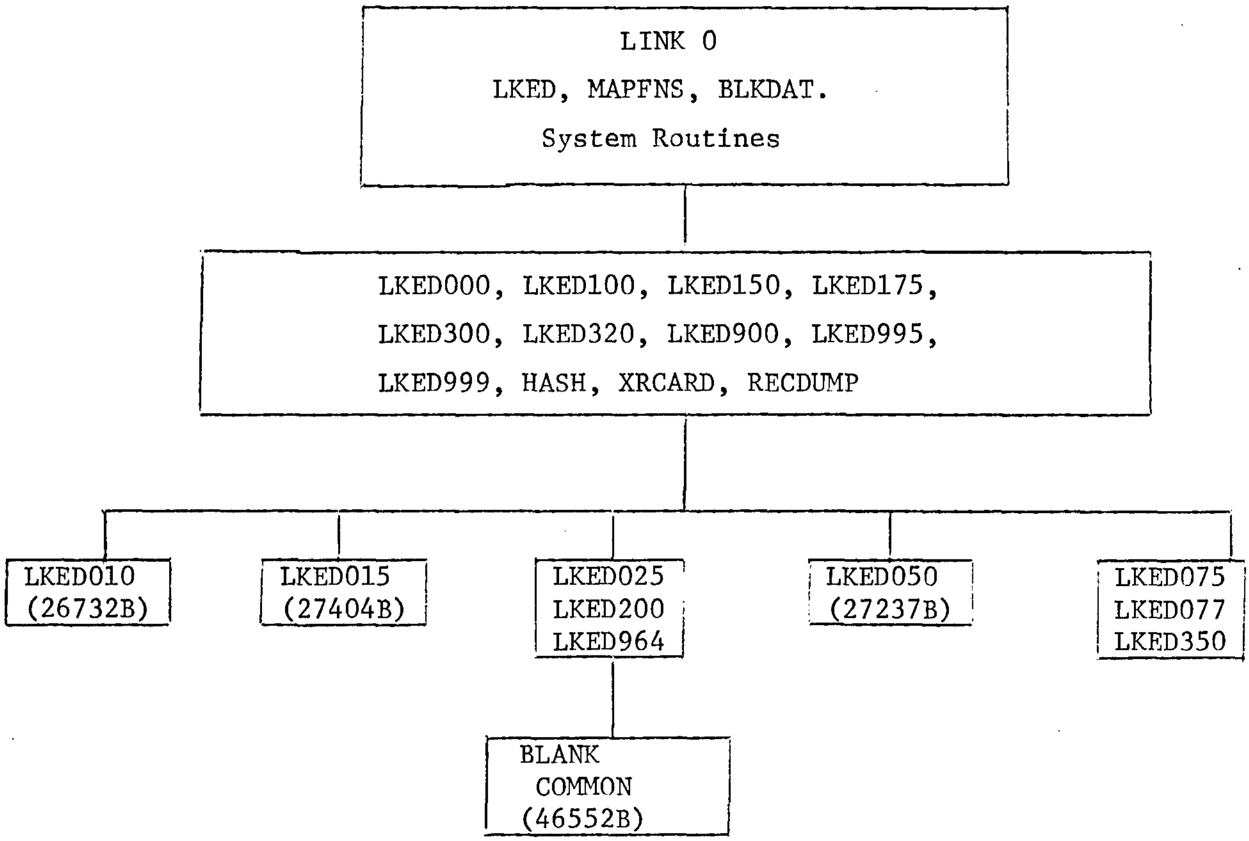


Figure 2 - Diagram of Link-Edited Version of the Linkage Editor

Scope Control Cards:

```
JOBCARD

CHARGE CARD

RFL,1300.

LABEL (TAPE,L=CARUCA1277,R,D=HI) (CA1277/NORING)

RFL,10000.

COPYBF, TAPE, LINKLIB.

COPYBF, TAPE, LINKEDT.

RETURN, TAPE.

RFL,70000.

NOREDUCE.

LINKEDT.
```

```
N S R D C   F T N - R U N  V E R S I O N  L I N K A G E   E D I T O R / L O A D E R    13.39.53.   10/26/72
            LEVEL 1.0 REVISIONS FOR FORTRAN EXTENDED COMPILER BY RJM/JMM
            CDC 6000 SERIES SCOPE 3 OPERATING SYSTEM / RUN-FTN COMPILERS

             LINKEDIT LET,OUTFILE=EDT(T),XREF,PARAM(7)=3
             LIBRARY LIB=LINKEDT

SEGMENT   1  LINK 0
             RENAME SYSTEM = SYSTEM.
             RENAME LKED000 = LINK
             INCLUDE LIB(LKED,MAPFNS)
             INCLUDE LIB(BLKDATA(LKEDC02))
             ENTRY LKED
             END
```

STORAGE MAP FOR LINK 0 13.39.53. 10/26/72

SEGMENT	NAME	ADDRESS	LENGTH	ENTRY-PT	ADDRESS	ENTRY-PT	ADDRESS	ENTRY-PT	ADDRESS	ENTRY-PT	ADDRESS	ENTRY-PT	ADDRESS
1	/LINK0$ /	000101	001273										
1	LKED	001375	004046	LKED	005415	INPUT#	001375	OUTPUT#	003402	TAPE5#	001375	TAPE6#	003402
1	MAPFNS	005444	000352	ANDF	005457	ORF	005463	COMPLF	005467	XORF	005445	RSHIFT	005505
				LSHIFT	005472	CORSZ	005451	CORWDS	005545	XSTORE	005520	XFETCH	005526
				LWORDS	005541	XJUMP	005552	ZAP	005555	LOCF	005564	FIELDLN	005567
				FLUSH	005605	RECOVRY	005611	TDATE	005654	DAYTIME	005675	KLOCK	005717
				LINK20.	005742	SET66	006004	INSTAL	006012	KLOCK1	005713	DMPXXXX	005746
				XCOMMON	005732								
1	/LKEDC02/	006017	000207										
1	/LKEDC03/	006227	000065										
1	/LKEDC04/	006315	000027										
1	/LKEDC05/	006345	000031										
1	/LKEDC07/	006377	000027										
1	/LKEDC08/	006427	000002										
1	SYSTEM$	006432	001011	FLSCM.	006436	FLLCM.	006437	Q8NTRY.	006440	END.	006456	EXIT$	006504
				STOP.	006510	ABNORM.	006517	SYSTEME	006535	SYSTEM$	006561	SYSTEM.	006565
				SYSTEM:	006622	SYS1:	007027	SYS2:	007023	LOT:	007235	ERRFLG.	007203
				DBGFET.	007255								
1	XLOADER	007444	000373	LOADER.	007567	LINK$	007444	LINK	007451	XTRACE	007747	XDUMP1	007702
				XDUMP	007706	COMPARE	007743	ABSENT.	010034				
1	CPC	010040	000242	CPC	010106	CPC02	010165	CPC03	010040	CPC04	010057	CPC999	010272
1	FTNFIX	010303	000030	SAVEALL	010303	SETB	010323	RESET	010315				
1	SIO$	010334	001406	CIO1.	011417	RCL1.	011430	DAT.	010336	SIO.CTL	010564	INITL.	010605
				SIO.	010655	SIO.END	011372	OPEN.	011436	RDPRU.	011511	BKSPRU.	011527
				ADVIN.	011534	POSFI.	011562	MVWDS.	011712	SYSERR.	011723		
1	OUTPTC$	011743	000074	OPUTCI.	011774	OUTPTC.	012003						
1	LINKERR	012040	000123	LINKERR	012042								
1	CORDUMP	012164	000362	CORDUMP	012166								
1	XIORTNS	012547	000347	XOPEN	012557	XCLOSE	012617	XEVICT	012631	REINDX	012636	XWRITE	012645
				READX1	013024	XREAD	012670	XREWIND	012723	XBKREC	012732	XFRDREC	012746
				XBKPREC	012741	XREQST	012753	WRITEX	012770	READX	013030		
1	KODER$	013117	001413	KODER.	013120								
1	IORANDM	014533	000203	IORR	014533	IORW	014546	IORRW	014573				
1	IO	014737	000213	IOREAD	014777	IOWRITE	015005	IOREWRT	015002	IOIO	015010	IOSAV	014750
				IOZZ	015143	IOZW	015147	D.BCDRD	014767	D.BCDWR	014773		
1	GETBA	015153	000017	GETBA	015154								
1	LOCF$	015173	000002	LOCF$	015173								
1	ACGOER$	015176	000013	ACGOER.	015177								

LAST ADDRESS IN LINK = 015210

REFERENCES FROM EACH SUBPROGRAM IN LINK 0 13.39.53. 10/26/72

SUBPROGRAM	ADDRESS	CALL	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION
LKED	001375	END.	004034										
		STOP.	004033										
		LINK	004031										
		FIELDLN	004027										
		RECOVRY	004024										
		SET66	004022										
		Q8NTRY.	004020										
MAPFNS	005444	SYSTEM$	000141										
		CPC	000126	000150	000213	000234	000256	000270					
		EXIT$	000202										
		LINK	000277										
		XDUMP1	000201										
		SETB	000055	000063									
SYSTEM$	006432	DAT.	000240	000271	000404	000410	000420	000442	000455				
		SIO.	000364	000365	000412	000414	000415	000362					
		ADVIN.	000535										
		RCL1.	000541										
		SIO.END	000536										
		INITL.	000207	000521									
		SIO.CTL	000523	000204	000166	000025							
XLOADER	007444	EXIT$	000372										
		OUTPTC.	000312	000340	000341								
		OPUTCI.	000311	000336									
		LINKERR	000120										
		CORDUMP	000262										
		CPC	000245	000063									
		DMPXXXX	000305										
		SETB	000243										
		READX1	000200	000054	000075	000037							
SIO$	010334	GETBA	000253										
OUTPTC$	011743	INITL.	000034										
		SIO.	000057										
		DAT.	000021	000053	000063								
		KODER.	000027	000041									
		SYSTEM$	000072										
		ABNORM.	000073										

LINKERR	012040	STOP.	000053	000044	000035	000026							
		ACGOER.	000013										
		OUTPTC.	000051	000050	000042	000041	000033	000032	000024	000023	000006		
		OPUTCI.	000046	000037	000030	000021	000005						
		OUTPUT#	000045	000036	000027	000020	000004						
CORDUMP	012164	LOCF$	000044	000031									
		XTRACE	000020										
		OUTPTC.	000241	000240	000223	000222	000207	000206	000177	000176	000174	000172	000162
			000161	000157	000155	000127	000126	000117	000116	000114	000112	000105	000104
			000102	000100	000023	000016							
		OPUTCI.	000236	000220	000204	000170	000153	000124	000110	000076	000022	000015	
		OUTPUT#	000234	000217	000202	000166	000152	000122	000106	000075	000021	000014	
		COMPARE	000057										
XIORTNS	012547	CPC	000035	000056	000063	000114	000155	000165	000173	000200	000212		
		IORR	000127										
		IORW	000104										
		IOREAD	000134										
		IOWRITE	000111										
		SETB	000070	000205	000222	000262							
		RESET	000116	000150	000042								
		SAVEALL	000011	000077	000122								
KODER$	013117	DAT.	000132	000720									
		ABNORM.	001106										
		SYSTEM:	001052	001070	001103	001105							
		ERRFLG.	001070	001102	001051								
IORANDM	014533	CPC	000020	000024	000045	000051							
		CPC02	000072										
		CPC999	000144	000150	000155	000161	000165	000171	000175	000201	000202		
		IOIO	000012										
		IOSAV	000001	000014	000041								
		IOZW	000141	000132									
		IOZZ	000142	000136									
IO	014737	CPC03	000012										
		CPC04	000204										
		CPC	000136										
		CPC999	000024										
ACGOER$	015176	ABNORM.	000004										
		SYSTEM:	000003										

REFERENCES TO EACH ENTRY POINT IN LINK 0 13.39.53. 10/26/72

ENTRY-PT	ADDRESS	CALL FROM	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION
LKED	005415	---NONE---											
INPUT#	001375	---NONE---											
OUTPUT#	003402	LINKERR	000045	000036	000027	000020	000004						
		CORDUMP	000234	000217	000202	000166	000152	000122	000106	000075	000021	000014	
TAPE5#	001375	---NONE---											
TAPE6#	003402	---NONE---											
ANDF	005457	---NONE---											
ORF	005463	---NONE---											
COMPLF	005467	---NONE---											
XORF	005445	---NONE---											
RSHIFT	005505	---NONE---											
LSHIFT	005472	---NONE---											
CORSZ	005451	---NONE---											
CORWDS	005545	---NONE---											
XSTORE	005520	---NONE---											
XFETCH	005526	---NONE---											
LWORDS	005541	---NONE---											
XJUMP	005552	---NONE---											
ZAP	005555	---NONE---											
LOCF	005564	---NONE---											
FIELDLN	005567	LKED	004027										
FLUSH	005605	---NONE---											
RECOVRY	005611	LKED	004024										
TDATE	005654	---NONE---											

DAYTIME	005675	---NONE---				
KLOCK	005717	---NONE---				
LINK20.	005742	---NONE---				
SET66	006004	LKED	004022			
INSTAL	006012	---NONE---				
KLOCK1	005713	---NONE---				
DMPXXXX	005746	XLOADER	000305			
XCOMMON	005732	---NONE---				
FLSCM.	006436	---NONE---				
FLLCM.	006437	---NONE---				
Q8NTRY.	006440	LKED	004020			
END.	006456	LKED	004034			
EXIT$	006504	MAPFNS	000202			
		XLOADER	000372			
STOP.	006510	LKED	004033			
		LINKERR	000053	000044	000035	000026
ABNORM.	006517	OUTPTC$	000073			
		KODER$	001106			
		ACGOER$	000004			
SYSTEME	006535	---NONE---				
SYSTEM$	006561	MAPFNS	000141			
SYSTEM.	006565	---NONE---				
SYSTEM:	006622	OUTPTC$	000072			
		KODER$	001052	001070	001103	001105
		ACGOER$	000003			
SYS1:	007027	---NONE---				
SYS2:	007023	---NONE---				
LOT:	007235	---NONE---				
ERRFLG;	007203	KODER$	001070	001102	001051	

DBGFET.	007255	---NONE---									
LOADER.	007567	---NONE---									
LINK$	007444	---NONE---									
LINK	007451	LKED	004031								
		MAPFNS	000277								
XTRACE	007747	CORDUMP	000026								
XDUMP1	007702	MAPFNS	000201								
XDUMP	007706	---NONE---									
COMPARE	007743	CORDUMP	000057								
ABSENT.	010034	---NONE---									
CPC	010106	MAPFNS	000126	000150	000213	000234	000256	000270			
		XLOADER	000245	000063							
		XIORTNS	000035	000056	000063	000114	000155	000165	000173	000200	000212
		IORANDM	000020	000024	000045	000051					
		IO	000136								
CPC02	010165	IORANDM	000072								
CPC03	010040	IO	000012								
CPC04	010057	IO	000204								
CPC999	010272	IORANDM	000144	000150	000155	000161	000165	000171	000175	000201	000202
		IO	000024								
SAVEALL	010303	XIORTNS	000011	000077	000122						
SETR	010323	MAPFNS	000055	000063							
		XLOADER	000243								
		XIORTNS	000070	000205	000222	000262					
RESET	010315	XIORTNS	000116	000150	000042						
CIO1.	011417	---NONE---									
RCL1.	011430	SYSTEM$	000541								
DAT.	010336	SYSTEM$	000240	000271	000404	000410	000420	000442	000455		
		OUTPTC$	000021	000053	000063						
		KODER$	000132	000720							

SIO.CTL	010564	SYSTEM$	000523	000204	000166	000025							
INITL.	010605	SYSTEM$	000207	000521									
		OUTPTC$	000034										
SIO.	010655	SYSTEM$	000364	000365	000412	000414	000415	000362					
		OUTPTC$	000057										
SIO.END	011372	SYSTEM$	000536										
OPEN.	011436	---NONE---											
RDPRU.	011511	---NONE---											
BKSPRU.	011527	---NONE---											
ADVIN.	011534	SYSTEM$	000535										
POSFI.	011562	---NONE---											
MVWDS.	011712	---NONE---											
SYSERR.	011723	---NONE---											
OPUTCI.	011774	XLOADER	000311	000336									
		LINKERR	000046	000037	000030	000021	000005						
		CORDUMP	000236	000220	000204	000170	000153	000124	000110	000076	000022	000015	
OUTPTC.	012003	XLOADER	000312	000340	000341								
		LINKERR	000051	000050	000042	000041	000033	000032	000024	000023	000006		
		CORDUMP	000241	000240	000223	000222	000207	000206	000177	000176	000174	000172	000162
			000161	000157	000155	000127	000126	000117	000116	000114	000112	000105	000104
			000102	000100	000023	000016							
LINKERR	012042	XLOADER	000120										
CORDUMP	012166	XLOADER	000262										
XOPEN	012557	---NONE---											
XCLOSE	012617	---NONE---											
XEVICT	012631	---NONE---											
REINDX	012636	---NONE---											
XWRITE	012645	---NONE---											
READX1	013024	XLOADER	000200	000054	000075	000037							
XREAD	012670	---NONE---											

XREWIND	012723	---NONE---			
XBKREC	012732	---NONE---			
XFRDREC	012746	---NONE---			
XBKPREC	012741	---NONE---			
XREQST	012753	---NONE---			
WRITEX	012770	---NONE---			
READX	013030	---NONE---			
KODER.	013120	OUTPTC$	000027	000041	
IORR	014533	XIORTNS	000127		
IORW	014546	XIORTNS	000104		
IORRW	014573	---NONE---			
IOREAD	014777	XIORTNS	000134		
IOWRITE	015035	XIORTNS	000111		
IOREWRT	015002	---NONE---			
IOIO	015010	IORANDM	000012		
IOSAV	014750	IORANDM	000001	000014	000041
IOZZ	015143	IORANDM	000142	000136	
IOZW	015147	IORANDM	000141	000132	
D.BCORD	014767	---NONE---			
D.BCOWR	014773	---NONE---			
GETBA	015154	SIO$	000253		
LOCF$	015173	CORDUMP	000044	000031	
ACGOER.	015177	LINKERR	000013		

```
N S R D C    F T N - R U N  V E R S I O N  L I N K A G E   E D I T O R / L O A D E R     13.39.53.   10/26/72
             LEVEL 1.0 REVISIONS FOR FORTRAN EXTENDED COMPILER BY RJM/JMM
             CDC 6000 SERIES SCOPE 3 OPERATING SYSTEM / RUN-FTN COMPILERS

SEGMENT   1    LINK 1
               RENAME SYSTEM = SYSTEM.
               INCLUDE LIB(LKED000,LKED100,LKED150,LKED175,LKED300,LKED320)
               INCLUDE LIB(LKED900,LKED995,LKED999,HASH,XRCARD,RECDUMP)
SEGMENT   2    OVERLAY A
               INCLUDE LIB(LKED010)
SEGMENT   3    OVERLAY A
               INCLUDE LIB(LKED015)
SEGMENT   4    OVERLAY A
               INCLUDE LIB(LKED025,LKED200,LKED964)
SEGMENT   5    OVERLAY LKEDCOR
               INSERT BLANK..
SEGMENT   6    OVERLAY A
               INCLUDE LIB(LKED050)
SEGMENT   7    OVERLAY A
               INCLUDE LIB(LKED075,LKED077,LKED350)
               ENTRY LKED000
               END
```

S T O R A G E   M A P   F O R   L I N K   1   13.39.53.   10/26/72

SEGMENT	NAME	ADDRESS	LENGTH	ENTRY-PT	ADDRESS	ENTRY-PT	ADDRESS	ENTRY-PT	ADDRESS	ENTRY-PT	ADDRESS	ENTRY-PT	ADDRESS
1	/SEGTAB$/	015212	000011										
1	LKED000	015224	002554	LKED000	015225								
1	LKED100	020001	000165	LKED100	020003								
1	LKED150	020167	000164	LKED150	020171								
1	LKED175	020354	000065	LKED175	020356								
1	LKED300	020442	000220	UNPKSYM	020443	PACKSYM	020460	PACK	020473	PACKMSK	020501	UNPK	020511
				UNPKMSK	020517	UNPK12	020526	PACK12	020537	SYMHASH	020552	CALLCHN	020565
				SEGPATH	020604	CONVERT	020616	RSHIFTX	020630	CHARTST	020634	UNPKXX	020643
				PACKXX	020654								
1	LKED320	020663	000345	UNPKCAL	020664	PACKCAL	020700	GETEXT	020713	STOEXT	020720	LKED990	020727
				NOW	020742	TEXTTAB	020760	FILLTAB	021013	LINKTB1	021054	LINKTB2	021072
				REPLTAB	021132	UNPK30	021170	UNPKID	021213	PACKDMP	021224		
1	LKED900	021231	000030	LKED900	021233								
1	LKED995	021262	000033	LKED995	021264								
1	LKED999	021316	000754	LKED999	021320								
1	HASH	022273	000024	HASH	022275								
1	XRCARD	022320	001544	XRCARD	022322								
1	RECDUMP	024065	000152	RECDUMP	024067								
1	/LKEDC01/	024240	000045										
1	/LKEDC06/	024306	000135										
1	REMARK$	024444	000030	REMARK$	024444								
1	INPUTS$	024475	000055	DECODI.	024503	DECODE.	024477						
1	INPUTC$	024553	000127	IPUTCI.	024614	INPUTC.	024637	EOF.CHK	024677				
1	DTOIE	024703	000021	DTOI.	024704	DTOI$.	024704						
1	ITOJE	024725	000016	ITOJ.	024731	ITOJ$.	024731						
1	KRAKER$	024744	001532	KRAKER.	024745	ERRSET$	025467						
1	/ENTAB$ /	026477	000025										
2	LKED010	026525	001123	LKED010	026526	LKED080	026757						
2	/ENTAB$ /	027651	000003										
3	LKED015	026525	000714	LKED015	026526								
3	/ENTAB$ /	027442	000003										
4	LKED025	026525	002712	LKED025	026526								
4	LKED200	031440	000235	LKED200	031442	LKED201	031456						
4	LKED964	031676	000666	LKED964	031700								
4	/ENTAB$ /	032565	000003										
5	/BLANK../	032571	014000										
6	LKED050	026525	000546	LKED050	026526								
7	LKED075	026525	002346	LKED075	026526								
7	LKED077	031074	001171	LKED077	031075								
7	LKED350	032266	000052	PACKXRF	032267	UNPKXRF	032304	UNPKEP	032323				
7	/ENTAB$ /	032341	000003										

LAST ADDRESS IN LINK = 046570

R E F E R E N C E S   F R O M   E A C H   S U B P R O G R A M   I N   L I N K   1   13.39.53.   10/26/72

SUBPROGRAM	ADDRESS	CALL	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION
XCLOSE	000000	PACK12	000107										
		UNPK12	000074										
		PACKMSK	000120	000053									
		LKED900	000032										
		SYMHASH	000013										
		HASH	000006										
		ANDF	000063										
LKED150	020167	PACKXX	000111										
		PACK12	000104	000055									
		PACKMSK	000100	000051	000046	000042	000034						
		UNPK12	000062	000014									
		UNPKXX	000007										
LKED175	020354	SYMHASH	000020										
		HASH	000013										
LKED300	020442	RESET	000137										
		SAVEALL	000124										
		SETB	000050	000056	000111	000040	000032						
LKED320	020663	CPC	000061	000063	000302								
		LKED995	000051										
		XDUMP1	000301										
		RESET	000243										
		SAVEALL	000210										
		SETB	000250	000131	000076								
LKED900	021231	STOP.	000012										
		OUTPTC.	000010	000007									
		OPUTCI.	000005										
		TAPE6#	000004										
LKED995	021262	LKED999	000012										
		OUTPTC.	000010	000007									
		OPUTCI.	000005										
		TAPE6#	000004										
LKED999	021316	UNPKCAL	000423										
		UNPKSYM	000354	000274									
		UNPK12	000325	000121	000062								
		OUTPTC.	000456	000455	000453	000451	000445	000443	000441	000437	000435	000433	000431
			000427	000412	000407	000406	000376	000375	000373	000371	000367	000342	000337
			000336	000314	000313	000311	000307	000305	000267	000264	000241	000240	000236
			000217	000177	000176	000152	000140	000137	000135	000133	000131	000127	000113
			000101	000100	000076	000074	000072	000055	000043	000042	000040	000026	000020
			000017										
		OPUTCI.	000425	000411	000404	000365	000341	000334	000303	000266	000263	000234	000216
			000174	000151	000125	000112	000070	000054	000036	000025	000015		
		TAPE6#	000424	000410	000402	000364	000340	000331	000300	000265	000260	000226	000213
			000171	000150	000123	000111	000065	000053	000033	000023	000014		
		UNPKMSK	000255	000167	000163								

HASH	022273	RSHIFTX	000006									
XRCARD	022320	OUTPTC.	000773	000766	000761	000760	000752	000751	000743	000736		
		OPUTCI.	000772	000765	000756	000747	000742	000735				
		OTOI$.	000731	000655								
		ITOJ$.	000033									
		ANDF	000333	000273	000112							
		COMPLF	000074	000056								
		ORF	000343	000304								
		LSHIFT	000337	000300	000072	000060						
		RSHIFT	000110	000062	000051	000042						
RECDUMP	024065	LOCF$	000056	000052								
		XDUMP	000062									
		XREAD	000037									
		OUTPTC.	000050	000026	000025	000023	000021	000017				
		OPUTCI.	000047	000015								
		OUTPUT#	000046	000014								
INPUTS$	024475	KRAKER.	000003	000020								
		DAT.	000007	000030	000037	000042						
		ABNORM.	000024									
		SYSTEM:	000023									
INPUTC$	024553	DAT.	000033	000104	000112	000120						
		SIO.	000114									
		INITL.	000043									
		KRAKER.	000066	000037	000062							
		SYSTEM:	000060									
		ABNORM.	000061									
		SYSERR.	000075									
KRAKER$	024744	DAT.	000126	000547	000550	000631	000637	001041				
		ABNORM.	001035	001150								
		SYSTEM:	000645	000662	000761	001006	001034	001066	001114	001132	001145	001147
		SYS1:	000575	000570	000763							
		SYS2:	000767	000752								
		LOT:	000771	000765								
		ERRFLG.	001065	000644	001113	000761	001132	001144	000661	001006		
		EOF.CHK	000215	000211								
PACK12	000000	PACKCAL	000146									
		PACK12	000133									
		UNPK12	000112									
		PACKMSK	000125	000105	000070							
		LKED900	000044									
		SEGPATH	000012									
		UNPKMSK	000101	000062								
		CALLCHN	000032									

```
LKED964   031676   UNPK30    000400  000354  000227
                   UNPKIO    000023
                   OUTPTC.   000434  000433  000431  000417  000416  000414  000412  000410  000370  000367  000365
                             000363  000350  000347  000345  000334  000333  000331  000322  000321  000317  000315
                             000313  000306  000305  000303  000301  000277  000272  000271  000267  000265  000263
                             000261  000257  000247  000246  000244  000242  000240  000220  000217  000215  000210
                             000204  000200  000167  000166  000164  000162  000160  000151  000150  000146  000144
                             000142  000140  000136  000120  000117  000115  000106  000105  000103  000101  000077
                             000063  000062  000060  000020
                   OPUTCI.   000427  000406  000361  000343  000327  000311  000275  000255  000236  000213  000174
                             000156  000134  000113  000075  000056  000017
                   TAPE6#    000426  000404  000360  000341  000326  000310  000274  000253  000235  000212  000171
                             000155  000131  000112  000071  000055  000014
                   UNPKMSK   000127
                   GETEXT    000403

LKED050   026525   OUTPTC.   000350  000343
                   OPUTCI.   000347  000342
                   TAPE6#    000346  000341
                   LKED100   000356  000321
                   LKED990   000362  000273  000264  000251
                   ACGOER.   000215
                   PACKMSK   000367  000337  000261  000237  000230  000156  000151  000111  000073  000034
                   UNPKSYM   000207  000021
                   UNPK12    000172  000123  000044  000010
                   UNPKMSK   000365  000335  000327  000310  000254  000175  000141  000055

LKED077   000000   UNPKXRF   000610
                   UNPKEP    000563
                   REINDX    000463  000451
                   XEVICT    000427
                   XREWIND   000424
                   PACK12    000414  000350
                   PACKXRF   000345  000265
                   UNPK12    000206
                   PACKMSK   000554  000551  000532  000525  000517  000232  000167  000130  000123  000116  000102
                   XREAD     000476  000466  000456  000242  000134  000045  000033
                   OUTPTC.   000663  000654  000650  000645  000640  000634  000631  000627  000625  000576  000575
                             000573  000436  000435  000433  000411  000405  000402  000375  000371  000366  000364
                             000362  000326  000322  000317  000312  000306  000303  000301  000277  000024  000023
                             000021
                   OPUTCI.   000662  000643  000623  000571  000431  000400  000360  000315  000275  000017
                   TAPE6#    000661  000642  000621  000567  000430  000377  000356  000314  000273  000016
                   SYMHASH   000510  000145  000060
                   HASH      000504  000141  000054
```

R E F E R E N C E S   T O   E A C H   E N T R Y   P O I N T   I N   L I N K 1   13.39.53.   10/26/72

ENTRY-PT	ADDRESS	CALL FROM	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION	LOCATION
OUTPUT#	003402	RECDUMP	000046	000014									
TAPE5#	001375	---NONE---											
TAPE6#	003402	LKED900	000004										
		LKED995	000004										
		LKED999	000424	000410	000402	000364	000340	000331	000300	000265	000260	000226	000213
			000171	000150	000123	000111	000065	000053	000033	000023	000014		
		LKED964	000426	000404	000360	000341	000326	000310	000274	000253	000235	000212	000171
			000155	000131	000112	000071	000055	000014					
		LKED050	000346	000341									
		LKED077	000661	000642	000621	000567	000430	000377	000356	000314	000273	000016	
ANDF	005457	XCLOSE	000063										
		XRCARD	000333	000273	000112								
ORF	005463	XRCARD	000343	000304									
COMPLF	005467	XRCARD	000074	000056									
RSHIFT	005505	XRCARD	000110	000062	000051	000042							
LSHIFT	005472	XRCARD	000337	000300	000072	000060							
CORSZ	005451	---NONE---											
CPC	010106	LKED320	000361	000063	000302								
STOP.	006510	LKED900	000012										
ABNORM.	006517	INPUTS$	000024										
		INPUTC$	000061										
		KRAKER$	001035	001150									
SYSTEM:	006622	INPUTS$	000023										
		INPUTC$	000060										
		KRAKER$	000645	000662	000761	001006	001034	001066	001114	001132	001145	001147	
SYS1:	007027	KRAKER$	000575	000570	000763								
SYS2:	007023	KRAKER$	000767	000752									
LOT:	007235	KRAKER$	000771	000765									
ERRFLG.	007203	KRAKER$	001065	000644	001113	000761	001132	001144	000661	001006			
XDUMP1	007702	LKED320	000301										
XDUMP	007706	RECDUMP	000062										

```
OPUTCI.    011774   LKED900  000005
                    LKED995  000005
                    LKED999  000425  000411  000404  000365  000341  000334  000303  000266  000263  000234  000216
                             000174  000151  000125  000112  000070  000054  000036  000025  000015
                    XRCARD   000772  000765  000756  000747  000742  000735
                    RECDUMP  000047  000015
                    LKED964  000427  000406  000361  000343  000327  000311  000275  000255  000236  000213  000174
                             000156  000134  000113  000075  000056  000017
                    LKED050  000347  000342
                    LKED077  000662  000643  000623  000571  000431  000400  000360  000315  000275  000017

OUTPTC.    012003   LKED900  000010  000007
                    LKED995  000010  000007
                    LKED999  000456  000455  000453  000451  000445  000443  000441  000437  000435  000433  000431
                             000427  000412  000407  000406  000376  000375  000373  000371  000367  000342  000337
                             000336  000314  000313  000311  000307  000305  000267  000264  000241  000240  000236
                             000217  000177  000176  000152  000140  000137  000135  000133  000131  000127  000113
                             000101  000100  000076  000074  000072  000055  000043  000042  000040  000026  000020
                             000017
                    XRCARD   000773  000766  000761  000760  000752  000751  000743  000736
                    RECDUMP  000050  000026  000025  000023  000021  000017
                    LKED964  000434  000433  000431  000417  000416  000414  000412  000410  000370  000367  000365
                             000363  000350  000347  000345  000334  000333  000331  000322  000321  000317  000315
                             000313  000306  000305  000303  000301  000277  000272  000271  000267  000265  000263
                             000261  000257  000247  000246  000244  000242  000240  000220  000217  000215  000210
                             000204  000200  000167  000166  000164  000162  000160  000151  000150  000146  000144
                             000142  000140  000136  000120  000117  000115  000106  000105  000103  000101  000077
                             000063  000062  000060  000020
                    LKED050  000350  000343
                    LKED077  000663  000654  000650  000645  000640  000634  000631  000627  000625  000576  000575
                             000573  000436  000435  000433  000411  000405  000402  000375  000371  000366  000364
                             000362  000326  000322  000317  000312  000306  000303  000301  000277  000024  000023
                             000021

XOPEN      012557 ---NONE---

XCLOSE     012617 ---NONE---

XEVICT     012631   LKED077  000427

REINDX     012636   LKED077  000463  000451

XWRITE     012645 ---NONE---

XREAD      012670   RECDUMP  000037
                    LKED077  000476  000466  000456  000242  000134  000045  000033

XREWIND    012723   LKED077  000424

WRITEX     012770 ---NONE---

READX      013030 ---NONE---
```

SAVEALL	010303	LKED300	000124					
		LKED320	000210					
SETB	010323	LKED300	000050	000056	000111	000040	000032	
		LKED320	000250	000131	000076			
RESET	010315	LKED300	000137					
		LKED320	000243					
DAT.	010336	INPUTS$	000007	000030	000037	000042		
		INPUTC$	000033	000104	000112	000120		
		KRAKER$	000126	000547	000550	000631	000637	001041
INITL.	010605	INPUTC$	000043					
STO.	010655	INPUTC$	000114					
SYSERR.	011723	INPUTC$	000075					
LOCF$	015173	RECDUMP	000056	000052				
ACGOER.	015177	LKED050	000215					
LKED000	015225	---NONE---						
	000000	---NONE---						
LKED077	031075	---NONE---						
PACKXRF	032267	LKED077	000345	000265				
UNPKXRF	032304	LKED077	000610					
UNPKEP	032323	LKED077	000563					

```
NSRDC   FTN-RUN VERSION LINKAGE  EDITOR/LOADER    13.39.53.   10/26/72
        LEVEL 1.0 REVISIONS FOR FORTRAN EXTENDED COMPILER BY RJM/JMM
        CDC 6000 SERIES SCOPE 3 OPERATING SYSTEM / RUN-FTN COMPILERS

        ENDLINKS

LINK  0 HAS BEEN WRITTEN ON EDT

LINK  1 HAS BEEN WRITTEN ON EDT
```

SUGGESTIONS FOR FURTHER IMPROVEMENTS

The following problem areas of the Linkage Editor should be examined in the future:

1. The Linkage Editor resolves externals only from those libraries listed on INCLUDE commands and from LINKLIB. The ability to concatenate several files and rename them LINKLIB with the LIBRARY command would be very helpful.

2. The Linkage Editor should be changed to allow both the file type (R) and XREF to be specified for the same job step. The problem appears to be that the return address to the entry point of Link 0 is not saved correctly.

3. An investigation should be conducted to determine the feasibility of dynamically allocating memory when segments are loaded.

4. "RANDOM CALL TO NONRANDOM FILE" errors occur when two or more Linkedit runs are made in the same job. This error apparently occurs because SYSUT1, SYSUT2, SYSUT3, and SYSLMOD are not closed between runs. This problem should be investigated and corrected.

ACKNOWLEDGMENT

The author wishes to thank Mr. James M. McKee (1844) for the extensive technical assistance he provided and for his help in defining the problem areas and the methods needed to accomplish the desired changes.

APPENDIX A

MODIFICATIONS TO THE LINKAGE EDITOR

The following paragraphs describe the various corrections and improvements made to the Linkage Editor:

1. The last card image on the original program library was an end-of-record card. This caused an error when compiling the last routine, XEOF. The card was deleted.

2. When a program containing a BLOCK DATA subroutine was edited, the following error message was written, even though no error existed.

---ERROR---ENTRY TABLE DOES NOT FOLLOW PIDL
TABLE IN SUBPROGRAM ________

This was a logic error in the Linkage Editor. BLOCK DATA subroutines do not contain ENTRY tables. This error was corrected by changing LKED025 so that it would branch around ENTRY table processing when a BLOCK DATA subroutine is being processed.

3. The end-of-card control character ("$") was not practical for FTN since many FTN routine names end in "$". The code checking for the character was deleted from XRCARD, and subsequently restored and changed to "*".

4. Code names for the various installations using the NASTRAN Linkage Editor were deleted from SET66 in MAPFNS. Now only the default code name "STANDARD" is accepted.

5. XLOADER is the program which handles the fetching and loading of each link as it is loaded. It was designed for RUN which passes argument addresses in the B registers. FTN passes the argument addresses in a list pointed to by register A1. In addition, register A0 must be preserved. The code in LOADER and LINK (both entry points to XLOADER) has been changed so that not only the B register but also the A0 and A1 registers are saved during the loading of a segment.

6. The code in XLOADER was changed to issue a memory macro call which returns the current field length.

7. XLOADER failed to set a return address in a link being loaded. Thus return could never be made from that link and the job would hang. Code was added to LINK to store the return address to enable return from the link to the calling segment.

8. REPLTAB, the routine to expand replication tables, did not comply with the specifications of REPL tables defined in the Scope reference manuals. The logic was changed to the following.

If LR $\neq$ SR, then return.

If DR = SR, then D = S.

If D = 0, then D = S+B.

If D = 0 and DR = 0, abort.

9. The Linkage Editor was changed to allow dynamic adjustment of the field length to that needed to load the current link. To obtain this dynamic allocation, include the following card in the LINK 0 control cards:

RENAME LINK = LINK$

If dynamic allocation is not desired, the NOREDUCE option must be used and the field length set at that needed to load the longest link.

10. The dayfile and header messages were changed to reflect changes in the Linkage Editor system.

11. A logic error limited BLOCK DATA subroutines to the definition of one and only one named common. The error was traced to REPLTAB and eliminated by returning to LKED075.

12. Unresolved external references occurred on the XBOOT step because not all needed routines were included on the list BOOTDKS. The needed routines were added to BOOTDKS.

13. The type of outfile to be generated was designated by a code character. In the original system these were C for COMMON (random file) and T for tape (sequential). These codes are ambiguous so the code S for sequential and R for random were added. Both codes are valid for each type of file.

14. REGION lines were printed when the options NOMAP and LET were selected concurrently. the code of LKED075 was changed to stop this error.

15. Automatic reduce could not be used with a random outfile because blank common had been dimensioned to one (1) word. The array size was increased to 200 words to eliminate the need for the NOREDUCE option.

16. When the options XREF and OUTFILE (R or C) are specified on the LINKEDIT card, the random outfile is incorrectly created. Code was added to flag this situation as an "error-exit" condition.

17. XBOOT was changed to check only the first six (6) characters when searching for "ATTACH", "CATLOG" or "CREATE" on an outfile execution card. This change was necessitated because INTERCOM appends a period to all commands.

18. To enable the Linkage Editor to be run on non-standard systems, all system routines which are needed for the execution of the bootstrap routine should be loaded with XBOOT. The Linkage Editor was changed to automatically load the needed system routines from LINKLIB.

19. The default value of PARAM (7) was changed to 3. This change prevents LKED077 from aborting when XREF is selected and PARAM (7) is not set.

The remaining updates were made to convert the Linkage Editor to FTN compilable code. A number of changes were made to FORTRAN routines, but the bulk of the work consisted of converting the COMPASS routines to correctly pick up the addresses of passed arguments.

APPENDIX B

DETAILS OF THE CDC 6400/6600 LINKAGE EDITOR

The following pages have been excerpted from the NASTRAN Programmer's Manual[1] and reproduced here for the user's convenience:

5.6 THE CDC 6400/6600 LINKAGE EDITOR

5.6.1 <u>Introduction</u>

5.6.1.1 Concept of the Linkage Editor

The linkage editor has been designed to provide an efficient and effective means of utilizing core storage for medium to large programs. The existing loader for the CDC 6400/6600 systems has the following disadvantages:

1. Only two levels of overlay are provided beyond the root segment.
2. An overlay segment must be <u>explicitly</u> called. Consequently, the overlay structure must be known when the program is coded.
3. An overlay segment may be entered at one point only. Consequently, downward calls are extremely limited.
4. No facility exists to explicitly position named common blocks.
5. Loading of overlay segments is accomplished from a sequential file, thus providing unnecessary search time.

The CDC 6400/6600 Linkage Editor in conjunction with its partner, the Segment Loader, overcomes these disadvantages in the following ways:

1. An unlimited number of overlay levels is provided.
2. The programmer describes the overlay structure to the linkage editor after the program is coded. The linkage editor provides <u>implicit</u> segment loading.
3. Complete communication between all levels of overlay is maintained.
4. Linkage editor control statements may be used to explicitly position subprograms and named common blocks.
5. The overlay segments are maintained in an indexed file. Consequently, every segment is immediately available to the segment loader.

As may be seen from Figure 1, the primary input sources to the linkage editor include:

1. Object decks (relocatable binary decks)
2. Control statements

3. A call library from which unsatisfied external references are resolved.

Another source of input (not shown in Figure 1) is a file containing executable links from a previous linkage editor run. This feature allows changes or additions of links while not altering previous links to which no changes are required.

The file produced by the linkage editor contains three portions:

1. A sequence of object decks suitable for loading by the CDC loader. The main program in this sequence, named XBØØT, reads the remainder of the file containing the executable links and writes it on the disk as an indexed file. XBØØT reads Link 0 into central memory and transfers control to the entry point which initiates execution of the problem program. This sequence of decks is terminated by a null record.

2. Three records:

(1) Link 0 directory record;

(2) Link 0 symbol dictionary containing entry points and common blocks in Link 0 and their associated addresses;

(3) Link 0 executable record.

3. A directory record for each succeeding link and one logical record per segment containing executable instructions and data.

This sequence of records is terminated by a directory record which contains the word ENDLINKS.

Link 0 remains in central memory at all times during program execution. Link 0 contains no overlay segments. The linkage editor supplies a routine named XLØADER when Link 0 is constructed. XLØADER accomplishes the loading of segments and links when requested. Segment load requests are supplied automatically by the linkage editor through tables called ENTAB$ (see section 5.6.3.2) which are written as a part of the text for each segment which may require additional segment loading. An additional table, SEGTAB$ (see section 5.6.3.2), which is constructed by the linkage editor as a part of the root segment of every link, is used by XLØADER to facilitate segment loading.

Major divisions of a program are links. Each link consists of a self-contained overlay structure and might be thought of as a complete program in itself. All routines in a link communicate freely with Link 0 routines. Consequently, Link 0 may be thought of as logically

belonging to every link. For many programs, a single link in addition to Link 0 will be sufficient. Because of its size, however, NASTRAN has been divided into 14 links.

5.6.1.2 Functions of the Linkage Editor

The basic function of the linkage editor is the linking of separately assembled or compiled subprograms into a link. The link is in a format suitable for loading and execution.

Although this linking or combining of subprograms is its primary function, the linkage editor also:

1. Incorporates subprograms from a library file to resolve undefined external references.
2. Constructs an overlay program in a format suitable for loading and execution.
3. Rearranges control sections and renames external references as directed by linkage editor control statements.
4. Reserves storage for common control sections generated by CØMPASS and FØRTRAN.
5. Provides processing options and diagnostic messages.

5.6.1.3 Subprogram Linkage

Processing by the linkage editor makes it possible for the programmer to divide his program into several subprograms which may be separately assembled or compiled. The linkage editor combines these subprograms into a link with contiguous storage addresses. The link is written in an indexed file. The linkage editor can process more than one link in a single job step. Each link is written with a unique link number.

5.6.1.4 Input Sources

Input to the linkage editor consists of one or more sequential files (libraries) containing subprograms in relocatable format as produced by CØMPASS or FØRTRAN, and linkage editor control statements contained in INPUT, the standard input file.

External references that are undefined after processing all subprograms cause the automatic library call mechanism to search for subprograms that will resolve the references. When these subprograms are found, they are processed by the linkage editor and become part of the link.

5.6.1.5 Programs in an Overlay Structure

To minimize main storage requirements, the programmer can organize his program into an overlay structure by dividing it into segments according to the functional relationshp of the subprograms. Two or more segments that need not be in main storage at the same time can be assigned the same storage addresses, and can be loaded at different times. The programmer uses linkage editor control statements to specify the relationship of segments within the overlay structure.

5.6.1.6 Options and Diagnostic Messages

The linkage editor can produce a storage map and a cross-reference table that show the arrangement of control sections in the link and how they communicate with each other. A list of the linkage editor control statements that were processed can be produced. Additionally, processing options that negate the effect of minor errors and specify the disposition of input and output files can be specified by the programmer.

Throughout processing by the linkage editor, errors and possible error conditions are printed. Serious errors cause a link not be written on the output file.

5.6.3 Designing an Overlay Program

5.6.3.1 Overlay Tree Structure

In order to place a program in an overlay structure, the programmer should be familiar with the following terms:

1. A control section consists of all instructions and data defined for a subprogram or a common block.

2. A segment is the smallest functional unit (one or more control sections) that can be loaded as one logical entity during program execution.

3. A path consists of a segment and all segments in the same region between it and the root segment (first segment). The root segment is a part of every path in every region. When a segment is in main storage, all segments in its path are also in main storage.

4. A region is a continguous area of main storage within which segments can be loaded independently of paths in other regions. An overlay program can be designed in single or multiple regions.

5. A link is a collection of one or more segments which comprise a logical subdivision of the program. Link 0 (consisting of one segment only) is in main storage at all times. It is the first link to receive control when execution of the program is initiated. The root segment of any other link resides in main storage at all times that that link is being executed. An overlay program must consist of at least one link other than Link 0.

6. A tree is the graphic representation that shows how segments can use main storage at different times. It does not imply the order of execution.

The design of an overlay program requires the organization of the control sections of the program in an overlay tree structure. The tree structure is developed considering:

1. The amount of available main storage.

2. The frequency of use of each control section.

3. The dependencies between control sections.

4. The manner in which control should pass within a path, from one path to another, and from one region to another.

When the programmer has determined the overlay structure for a program, he prepares ØVERLAY, INSERT and REGIØN statements that will segment the program in that manner. The use of these control statements is described in section 5.6.4.

5.6.3.2 Overlay Characteristics

During execution of an overlay program, the segment loader uses tables that were generated by the linkage editor and incorporated into the text of applicable segments. Since these tables are an integral part of the program, their size must be considered when planning the use of available main storage. These tables are described as follows.

1. <u>Input/Output Control Table</u>

   There is one Input/Output Control Table (LINKO$) in the root segment of Link 0 only which contains a File Environment Table (FET), a circular buffer, a master index and a subindex. The LINKO$ table is used by the segment loader to read requested segments into central memory. LINKO$ is the first control section in Link 0. Its size is determined as follows:

   $$\text{Length in words} = \text{PARAM(1)} + \text{PARAM(4)} + \text{PARAM(5)} + 4 \ .$$

   Section 5.6.4.2 contains definitions of the parameters.

2. <u>Segment Table</u>

   There is one Segment Table (SEGTAB$) in the root segment of each link except Link 0. The segment table is used to keep track of: (1) the relationship of the segments in the program; (2) which segments are in main storage or scheduled to be loaded; (3) the main storage address and length of each segment; and (4) the entry address of the link.

   SEGTAB$ is the first control section in the root segment of each link. Its size is determined as follows:

   $$\text{Length in words} = n + 2,$$

   where n is the number of segments in the link.

3. <u>Entry Table</u>

   There can be an Entry Table (ENTAB$) in each segment of the program. The loader

uses the entry table to determine the segment to be loaded when an external reference is made to a segment not in the path.

An entry table may be produced as the last control section of a segment. An ENTAB$ entry is created for a symbol to which control is to be passed. The symbol is defined in a segment not in the path. The size of ENTAB$ is determined as follows:

$$\text{Length in words} = 3n + \sum_{i=1}^{n} \delta_i,$$

where n is the number of unique external references not in the path and $\delta_i = \text{MAX}(m_i-6,0)$, m_i = number of arguments for each external reference not in the path.

4. Dump Control Word

In the text produced by the linkage editor for each segment, a uniquely formatted word which identifies the control section is written immediately prior to each control section. This word is recognized by the storage dump routine XDUMP in order to produce relative addresses for each control section.

5.6.3.3 Overlay Communication

There are two ways in which the programmer can have his program request the overlay facilities of the segment loader:

1. By a CALL statement (FØRTRAN language) or RJ instruction (CØMPASS language) which causes a segment to be loaded and control to be passed to the symbol defined in that segment.

2. By a CALL LINK(N) (FØRTRAN language) or the equivalent in the CØMPASS language, where N is the link number, which causes segment one (the root segment) of the requested link to be loaded and control to be passed to the symbol named on the linkage editor control statement ENTRY.

5.6.3.4 Reserving Storage

In FØRTRAN and CØMPASS the programmer can create control sections that reserve main storage areas containing no data or instructions. Referred to as "common", these control sections are produced by the language translator. These common areas are either named or blank (unnamed).

During processing, the linkage editor collects these common areas. If more than one blank common area is found, the largest blank common area is contained in the link. If two or more

common areas have the same name, the largest common area having that name is reserved in the link. All references to a common area (named or blank) refer to the largest area defined. This largest area is the one which is retained.

If the linkage editor encounters data or text for the same common area in more than one subprogram, only data from the first subprogram encountered are retained and a diagnostic message is generated for any subsequent data definitions.

When object decks which reference common areas are to be placed in an overlay structure, the linkage editor automatically "promotes" the common areas to the root segment (unless otherwise directed by an INSERT control statement, see section 5.6.4.8). The position of a promoted common area in relation to other control sections in the root segment is generally unpredictable.

<u>Note</u>: Blank common is treated by the linkage editor as a named common block with the special name BLANK.. and is listed on the storage map with this name. Consequently, it is possible to position this control section with the statement INSERT BLANK...

5.6.3.5 Processing Options

1. <u>List of control statements</u>

The linkage editor automatically produces a listing of all control statements unless the programmer selects the NØLIST option in the LINKEDIT statement (see section 5.6.4.2). In the latter case, only the LINKEDIT, LIBRARY and ENDLINKS statements are listed (see sections 5.6.4.2, 5.6.4.3 and 5.6.4.12 respectively for details).

2. <u>Storage map and cross-reference table</u>

The linkage editor automatically produces a storage map of each link unless the programmer selects the NØMAP option in the LINKEDIT statement. For each segment, the storage map lists the control sections in ascending order according to their assigned address. Included with each control section is a list of all entry point names and assigned addresses.

When the XREF option in the LINKEDIT statement is specified, the linkage editor produces a table of all references to each entry point in the link. Additional options (PARAM(7) parameter, see section 5.6.4.2) allow the table to be extended to include all references from the link to LINK 0 entry points and an additional table of all external references from each subprogram to be produced.

The NØMAP and XREF options are mutually exclusive. Therefore, if XREF is selected, NØMAP is ignored and a storage map is produced.

3. <u>The LET option</u>

When the LET option of the LINKEDIT statement is selected, the linkage editor disregards all errors except two and writes the link on the output file. The two errors which preclude the link from being written are: (1) an undefined entry point to the link; and (2) insufficient storage space to form the link to be written.

5.6.4 Linkage Editor Control Statements

5.6.4.1 General Statement Format

All linkage editor control statements are coded from the following possible forms:

operation	operand
VERB	a, b(c), KEYWØRD, KEYWØRD = a, KEYWØRD = b(c),
	KEYWØRD(i) = n, a = a, b(c) = a,n

where

a is an unsubscripted symbol,

b is a subscripted symbol,

c is a subscript symbol,

KEYWØRD is an explicit name or option,

i is an integer subscript,

n is an integer value.

The operation field must contain the name of the operation to be performed. The operand field must contain one or more symbols or subscripted symbols (except REGIØN, END and ENDLINKS which have no operands). Operands in the operand field are separated by a comma or blank (or both). Two or more symbols within parentheses are similarly separated. A keyword must be written exactly as shown.

The operation field begins with the first nonblank column on the card. The operand field is separated from the operation field by at least one blank column.

The LINKEDIT and LIBRARY control statements may be continued on subsequent cards by coding a comma as the last nonblank column. The continuation begins with the first nonblank column of the succeeding card. These two control statements are the only ones which may be continued.

5.6.4.2 The LINKEDIT Statement

The LINKEDIT statement specifies input and output file names and status, processing options and size characteristics of the link(s) to be link-edited.

5.6.6 Storage Requirements for the Linkage Editor

Figure 5 illustrates the layout of core storage for the linkage editor. For the discussion below, it is assumed that the linkage editor has not itself been link-edited. A link-edited version of the linkage editor is available. A memory saving of approximately $4000_{10}(10000_8)$ words results.

The principal open-ended table is the Symbol Chain Table. A three-word entry is created in this table for each subprogram name, entry point, common block and unique external reference not in the path. For a link other than Link 0, a three-word entry for each entry point and common block in Link 0 is also created. A conservative estimate for the requirements of this table is as follows:

Link 0: length in words = 4* (no. of entry points + common blocks),

Link $\neq$ 0: length in words = 6* (no. of entry points + common blocks)
+3* (no. of entry points + common blocks in Link 0).

The largest table is likely to be the Working Storage Table. It must hold all instructions and data for the largest control section for which text is defined. If this figure is not known, a linkage editor run can be made. The storage map will be printed even if the link is not written. A scan of the lengths listed (in octal) will identify the largest control section. Note that common blocks for which no data are defined are not to be used in defining the maximum.

Field length for the linkage editor may be estimated from the following:

$$\text{field length}_{10} = 15000 + \text{MAX}(10*N,2000) + \text{MAX}(T,2000) + 3*\text{PARAM}(1)$$

where

N = number of subprograms defined on INCLUDE statements,

T = length of largest subprogram or common block for which instructions or data are defined,

and

PARAM(1) is defined in section 5.6.4.2.

If default values for the linkage editor are used, a program of less than 200 decks would require a field length of $23{,}600_{10} \approx 60{,}000_8$.

Efficiency of the linkage editor may be improved by increasing the buffer size (PARAM(1)). For NASTRAN, PARAM(1) = 2080 is used. Additionally, one deck requires $16{,}000_{10}$ words of text storage (PARAM(6) = 16000). Consequently, for a link of 300 decks, the field length works out as

$$\text{field length}_{10} = 15000 + 3000 + 16000 + 6240 = 40240_{10} \approx 120000_8$$

Address	Contents	Size
0	Instructions and Data	
$\approx 14000_{10}$	$Buffer_1$	PARAM(1)
	$Buffer_2$	PARAM(1)
	$Buffer_3$	PARAM(1)
	Master Index	PARAM(4)
	Segment Index	PARAM(5)
	Library Index	No. of decks in all Libraries ($\leq$ PARAM(6))
	Names Table	No. of decks in all Libraries ($\leq$ PARAM(6))
	Entry Point Table	No. of entry points in LINKLIB ($\approx$ 200)
	Library Table	No. of libraries
	Region Definition Table	No. of regions + 1
	Segment Definition Table	No. of segments + 1 ($\leq$ PARAM(4) + 1)
	Segment Chains Table	No. of segments + 1 ($\leq$ PARAM(4) + 1)
	Rename Table	3*(no. of RENAME statements)
	Symbol Chain Table	Remaining storage
Field Length	Working Storage	PARAM(3) + PARAM(6)

Figure 5. Layout of core storage for the linkage editor.

7.2.1 Introduction

7.2.1.1 Purpose of the Linkage Editor

The linkage editor is a service program designed to be used in association with the RUN compiler to prepare an executable program from symbolic language programs written in FØRTRAN and CØMPASS. Linkage editor processing is a necessary step between source program compilation and object program execution.

Linkage editor processing allows the programmer to divide his program into several parts, each containing one or more control sections. Each part may then be coded in the programming language best suited to it and may then be separately assembled or compiled.

The primary purpose of the linkage editor is to combine and link object decks (the output of the RUN compiler) into a program in which all cross references between control sections are resolved as if they had been assembled or compiled as one program. The program produced by the linkage editor consits of executable machine language code in a format that can be loaded into main storage by the bootstrap program (see section 7.2.1.4.7) and segment loader (see section 7.2.1.4.8).

The main design objective of the linkage editor/loader is to efficiently process and execute unusually large programs that require extensive segmentation (a feature entirely lacking in the existing CDC loader).

In addition to combining and linkage object decks, the linkage editor performs the following functions:

1. Library Call Processing. If unresolved external references remain after the linkage editor processes all input to it, an automatic library call feature retrieves subprograms required to resolve the references.

2. Program Modification. Control sections can be rearranged during linkage editor processing as directed by linkage editor control statements. Common control sections are collected. References to entry points can be altered by control statements.

3. Overlay Processing. The linkage editor prepares programs for overlay by inserting tables (SEGTAB$, ENTAB$, see section 7.2.2.7) to be used by the segment loader during execution.

7.2.1.2 Relationship to the SCØPE Operating System

The linkage editor is not an integral part of the SCØPE operating system. As a result, it is executed as a normal "user" program, i.e., using the facilities of the CDC loader.

The object decks that comprise the linkage editor exist as a card, tape, or disk file and the linkage editor is executed as a normal job step.

The executable program produced by the linkage editor may be in the form of a sequential file on tape or disk or an indexed (random) file on disk. In either case, the initial records of the file contain object decks that comprise the bootstrap program loads the initial portion (Link 0) of the executable program into main storage and optionally writes the remaining links of the executable program. Thereafter, all loading of additional segments of the executable program is controlled by the segment loader which was included in Link 0 by the linkage editor.

In the Level 2.0 version of the linkage editor (the current version), processing is limited to object decks produced by the RUN compiler because of linkage conventions established by that compiler. Reasonably extensive modification of the linkage editor/loader and LINKLIB (see below) is required to process object decks produced by the FTN compiler.

Associated with SCØPE and the RUN Compiler are a number of subprograms which accomplish the primary interface between the user and the resident monitor. These subprograms are a principal input to the linkage editor and reside on a file named LINKLIB. Since the linkage editor is not an integral part of SCØPE, modification of the LINKLIB subprograms is not automatically accomplished with SCØPE updates and remains a maintenance task at each installation.

Linkage editor processing and subsequent execution time loading is dependent on the file concepts and operations as defined and supported in SCØPE 3.1. In particular, changes to the subfields of the File Environment Table (FET) or changes to the object deck format are likely to require modification to the linkage editor and segment loader code.

7.2.1.3 General Description

Input to the linkage editor consists of: a) one or more sequential files (libraries) containing subprograms in relocatable format (object decks) as produced by the RUN compiler, and

b) linkage editor control statements contained in INPUT, the standard input file. The primary function of the linkage editor is to combine these subprograms, in accordance with the requirements stated on the control statements, into a machine-language program suitable for loading into main storage and executing. External references that are undefined after processing all subprograms cause the automatic call mechanism to search for subprograms that will resolve the references. When these subprograms are found, they become part of the executable program.

To produce an executable program, the linkage editor:

1. Assigns relative main storage addresses to the control sections to be included in the program.

2. Resolves references between control sections (translates symbolic references to relative main storage addresses)

3. Collects common sections and assigns a single relative machine address to all sections of the same name. The length of the common section is taken to be the longest length of any individual section.

Figure 1 illustrates an example of linkage editor processing. The executable program produced by the linkage editor contains three portions:

1. A sequence of object decks suitable for loading by the CDC loader. The main program in this sequence, named XBØØT (see section 7.2.2.9), reads the remainder of the program and writes it on the disk as an indexed file (unless the program is already an indexed file). XBØØT reads Link 0 in main storage and passes control to the entry point which initiates execution of the problem program.

2. A sequence of three records which defines Link 0 - a directory record, a symbol dictionary record, and the executable machine language code:

3. A sequence of records for each of the additional links - one directory record per link plus one record containing executable machine language code for each segment in the link.

Link 0 remains in main storage at all times during program execution. Link 0 contains no overlay segments. The linkage editor supplies the segment loader (named XLØADER, see section 7.2.2.10) when Link 0 is constructed. XLØADER accomplishes the loading of segments and links

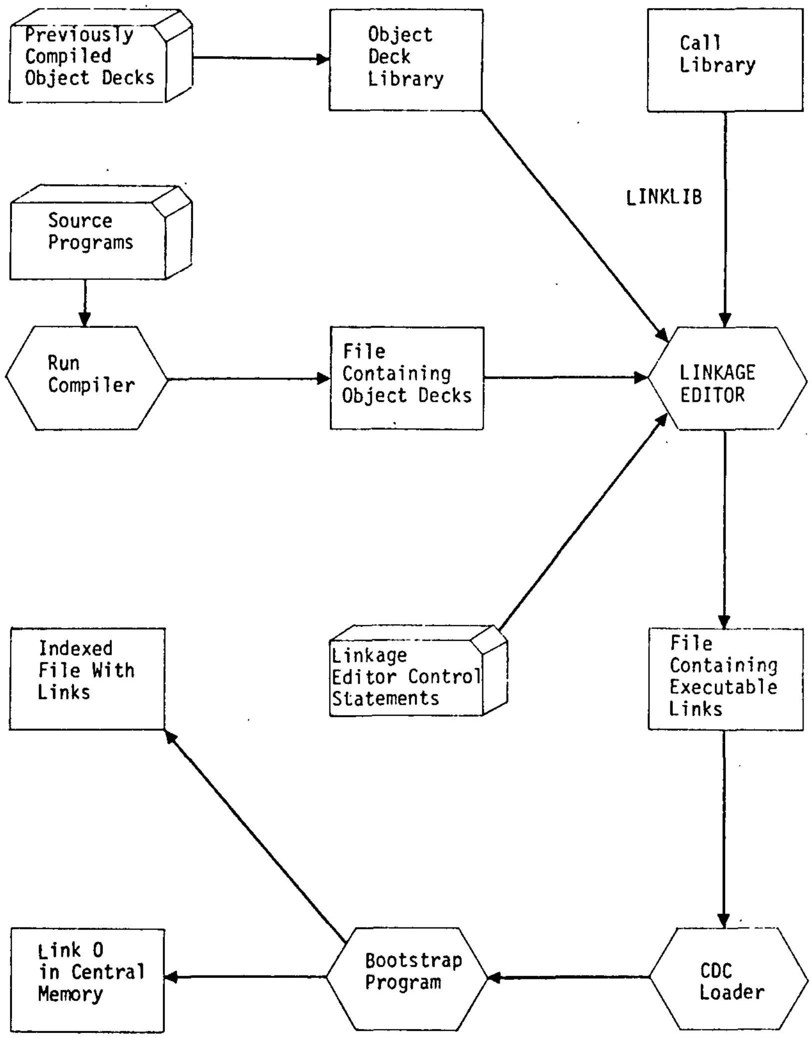


Figure 1. Linkage editor processing.

7.2-4a (6/1/71)

when requested. Segment load requests are supplied automatically by the linkage editor through tables called ENTAB$ (see Figure 29) which are written as a part of the text (instructions and data) for each segment which may require additional segment loading. An additional table, SEGTAB$ (see Figure 28) which is constructed by the linkage editor as a part of the root segment of every link is used by XLØADER to facilitate segment loading.

Major divisions of a program are links. Each link consists of self-contained overlay structure and might be thought of as a complete program in itself. All routines in a link communicate freely with Link 0 routines. Consequently, Link 0 may be thought of as logically belonging to every link.

7.2.1.4 Major Divisions of the Linkage Editor

7.2.1.4.1 Initial Processing

Initial processing begins when the linkage editor receives control from the CDC loader. After control is received, the following functions are performed:

1. The LINKEDIT card is read, echoed, and converted. Parameters are set based on options selected.

2. Initial allocation of working storage and buffers is made.

3. If the program from a previous linkage editor run is present as a sequential file (INFILE), it is read and written as an indexed file.

4. Each file named on the LIBRARY card is read. Each deck is written on a local disk file named SYSUT2 (indexed file). Subprogram names are saved in a main storage table. For the file named LINKLIB, each of the entry point names is saved in main storage.

7.2.1.4.2 Control Statement Processing

For a link, cards from LINK through END are read and converted. Two passes are made. On the first pass, each card is checked for proper format, content, and order (if important). Various counts are accumulated such as the number of segments, number of regions, number of RENAME cards, etc. The control statements are echoed on ØUTPUT unless this option is suppressed. At the end of the first pass, allocation of working storage is completed. If the currently processed link is not Link 0, the dictionary defining entry point and common block names and address

for Link 0 is read, and entries are made in the General Table (see section 7.2.2.1.9) for each Link 0 name and address.

On the second pass of the control statements, each statement (having been saved in main storage during the first pass) is again converted, and entries are made in various tables depending on the control statement and its contents.

Following the second pass of the control statements, control is passed to LKED025 (see Figure 35, section 7.2.3) to read each of the object decks named on INCLUDE statements plus those subprograms required to satisfy undefined external references.

7.2.1.4.3 Object Deck Processing

The list of subprogram names in each of the named libraries is scanned. For each subprogram which is marked for inclusion, the following processing occurs:

1. The deck is read from SYSUT2.
2. Subprogram (or common block) length is entered in the General Table (GT).
3. Each common block referenced by the subprogram is entered into the GT (if not already present), and the length field is updated. If text (data) for the common block exists, a reference to the defining subprogram is noted.
4. An entry in the GT is created for each entry point of the subprogram. The relative address of the entry point is saved. The number of arguments associated with each entry point is found by searching the TEXT tables (see section 7.2.5) for the conventional identification word. If not found, less than seven arguments is assumed.
5. The LINK table is processed. For each external reference by the subprogram, the GT is checked for an existing entry. If present, a path analysis is made. If the call is not in the path, a call chain entry is created in the GT. If the entry is not present, an entry in the GT is created and a call chain entry is created.

When all object decks have been processed, the automatic call logic is invoked. For each undefined external reference, the list of entry points to LINKLIB is searched. If found, the corresponding subprogram from LINKLIB is included. If not found, an error message is issued.

When all object decks from LINKLIB have been processed, a pass through each of the entries in the GT is made and various checks are made. Call chains are checked, and entries now resolved (in the path) are removed. Remaining entries in the call chains will require facilities of the segment loader, and these entries will form the ENTAB$ tables.

At this point, all information is available to perform assignment of final addresses for the program. Control is passed to LKED050 (see Figure 36, section 7.2.3) for this task.

7.2.1.4.4 Address Assignment Processing

The program computes final storage addresses for all subprograms, entry points, and common blocks in the program by executing the following steps:

1. Lengths for each segment are computed by summing the lengths of each entry (subprogram or common block) in the segment. This information is stored in the Segment Definition Table (see section 7.2.2.1.7).

2. The lengths for each region are computed by finding the longest path in the region and summing the length of all segments in that path.

3. Region lengths are converted to region addresses by summing the region lengths. This information is stored in the Region Definition Table (see section 7.2.2.1.5).

4. Segment addresses are computed by following the paths in each region and summing the previous segment lengths.

5. Finally, addresses for each entry in each segment are computed by tracing the order of each entry in the segment and summing lengths of previous entries.

7.2.1.4.5 Relocation Processing

The final phase for each link consists of building the executable machine language code, performing all necessary relocation of relative addresses.

This is accomplished by executing the following steps:

1. If the current link is Link 0, object decks defining the bootstrap program are copied from LINKLIB to the executable program file (either SYSUT1 or ØUTFILE). A directory record containing link number, number of entries in the Link 0 dictionary, and total length of the

link is written followed by the Link 0 dictionary defining each of the entry points and common blocks and their addresses in the link.

2. If the current link is not Link 0, a directory record containing link number, number of segments, and total length of the link is written as in 1. above.

3. The first entry in the root segment of each link is a table (LINKO$ for Link 0 and SEGTAB$ for any other link). This table is built and written.

4. Executable machine language code is built and written one logical record per segment. Each entry (subprogram or common block) in each segment is examined. If text (for a subprogram) or data (for a common block) is defined for the entry, the object deck containing the text is read from SYSUT2. Address relocation defined in TEXT, FILL, LINK, and REPL tables (see section 7.2.5) is performed, and the relocated text for the entry is written. If no text is defined for the entry, zero words are written.

5. As the relocation of text is being performed, the storage map is printed on ØUTPUT unless NØMAP was selected.

6. Finally, if an ENTAB$ table is defined for the segment, the text for this table is assembled and written as the last entry for the segment.

7. When all segments for the link are complete, the XREF option on the LINKEDIT card (see section 5.6.4.2) is tested. If selected, LKED077 (Figure 37, section 7.2.3) is called to produce a listing of all cross references in the link.

7.2.1.4.6 Final Processing

When processing for all links is complete (the ENDLINKS card has been read from INPUT), the status of ØUTFILE is tested. If ØUTFILE = name(C) was coded, no further processing is required. Otherwise, the executable program exists as a local indexed file (SYSUT1) and it is necessary to write it as a sequential file on the user-requested file. This is accomplished by LKED080 (Figure 38, section 7.2.3). When the link has been copied to ØUTFILE, a message is written on ØUTPUT indicating the event.

7.2.1.4.7 The Bootstrap Program

The bootstrap program is a computer program made up of relocatable routines which are appended by the linkage editor to the beginning of the absolute output of the linkage editor. These routines consist of: a) a dummy Block Data subprogram containing one labeled common block of a length sufficient to hold Link 0; b) the bootstrap program driving routine, XBØØT; c) an input/output utility routine XIØRTNS; and d) MAPFNS, a routine containing miscellaneous utility routines for bit manipulation, field length determination, etc.

The bootstrap program is employed to permit the execution of the absolute output of the linkage editor in a way that requires no special handling of the job and allows the job to appear as any other batch job. It is a small program, loaded by the CDC loader which if necessary reads and outputs to the disk the sequential linkage editor output in a direct access (random) format. The bootstrap program also reads into the locations 77_8+1 through 77_8+N Link 0 (N being its length). This core space is available because the CDC loader has placed the dummy Block Data subprogram there.

Having completed its function, the bootstrap program calls CØMPASS routine XJUMP in, MAPFNS which directs the central processor to jump to location 101_8 in the jobs core, which is in Supermain, and execution then continues from there. Figure 2 illustrates core through the bootstrap process. It should be noted that for the completion of this particular job step, execution of the bootstrap program is no longer required, nor is it available.

7.2.1.4.8 The Segment Loader

The bootstrap program is actually the initial loader of absolute object code as produced by the linkage editor. It does in fact load "Supermain," Link 0. After the bootstrap program directs the central processor to branch into Supermain, and execution proceeds from there, any calls for the loading of a link's root segment, results in an automatic transfer into the segment loader to the entry point LINK. Similarly, any calls to a segment lower in a tree or in another region results in an automatic call into the segment loader to the entry point LØADER.. This type of "downward" call is forced through an entry table ENTAB$ (see section 7.2.2.7) before reaching the segment loader at entry point LØADER..

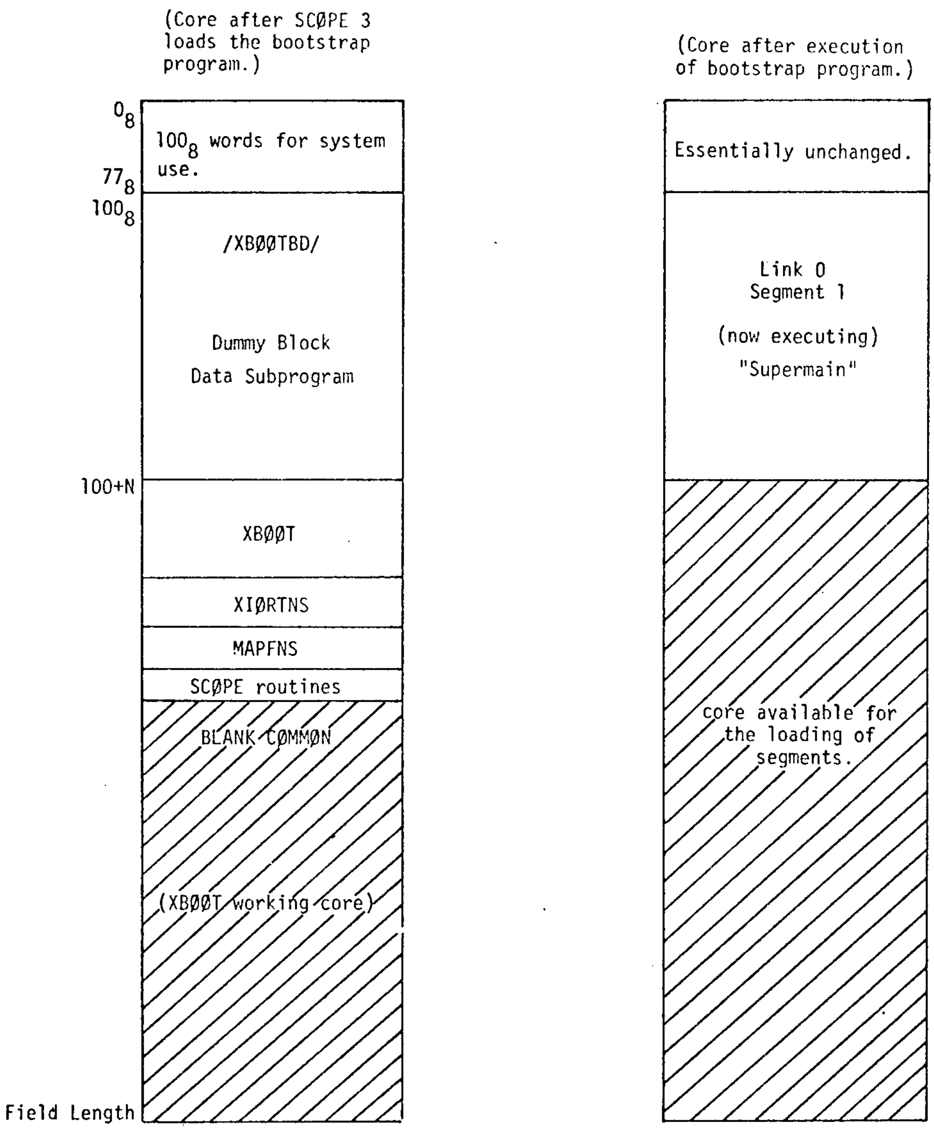


Figure 2. Core before and after execution of the bootstrap program.

7.2-9a (6/1/71)

Calls made to LINK from any segment, anywhere in core, result in the segment loader first checking the link number for legitimacy. The indexes of relative disk addresses for the segments of the link desired is then read from the disk. A link directory is then read from the disk and further legitimacy checks are made along with a check to insure that sufficient core is available for the loading of the lowest segment of the link.

After successfully completing these tasks, the root segment of the new link is read into core, and a branch is made to its entry point and execution of the program continues.

Downward calls reaching the entry point LØADER. via an ENTAB$ table result in a series of conditional events by the segment loader. The loader first checks to see if the segment to which the call is directed is in core. If the segment is not in core, it is loaded along with any segments above and in its path as required. Once the segment is determined to be in core, any argument addresses over six (which are assigned to B registers B1 through B6 by the RUN compiler generated code) are moved from the ENTAB$ entry and placed in the actual subroutine being called along with the actual branch return. A jump is then made to the desired entry point to complete the automatic loading process. Returns from any called control section are always made directly to the point from which the call was made.

7.2.1.5 Linkage Editor Files

7.2.1.5.1 Input Files

There are three types of files that may be input to the linkage editor. They are:

1. Libraries. All object decks that are to be processed by the linkage editor are contained in libraries. A library is defined to be a sequential file (which may reside on tape or disk) consisting of one or more logical records with one object deck per logical record. The names of the library files are defined on the LIBRARY control statement (see section 5.6.4.2). A file named LINKLIB must always exist for linkage editor processing. LINKLIB contains object decks for automatic library call plus object decks which are required in constructing the initial load portion (bootstrap program) of the executable program. There is no theoretical limit to the number of libraries which may be defined for linkage editor processing. Subprograms of the same name may appear in more than one library or even in the same library. In the latter case, the first such subprogram is included.

2. Control statements. Statements which direct and control processing by the linkage editor are contained as a single logical record on the file named INPUT. INPUT must be positioned to the logical record containing the control statements prior to executing the linkage editor. For a complete description of the linkage editor control statements, see section 5.6.4.

3. Previously link-edited links. This input source is optional and is required only if the user desires to modify an existing link (other than Link 0) or add a new link to the program. The name and status of this file is defined by the INFILE keyword on the LINKEDIT control statement (see section 5.6.4.2). It may be a sequential file on tape or disk or an indexed file on disk.

7.2.1.5.2 Local Files

These may be one, two or three local files generated by the linkage editor during processing. A file named SYSUT2 is always generated. It is an indexed file and contains all object decks from all defined libraries (including LINKLIB). When the file is being generated, a directory of subprogram names as well as a list of all entry points in LINKLIB is extracted and maintained in working storage. If either INFILE or ØUTFILE is declared as a common (indexed) file, then a second local file does not exist (note that if both INFILE and ØUTFILE are declared common files, they must be the same file). Otherwise, a local file named SYSUT1 is generated as an indexed file to contain each of the links as they are constructed. If the XREF option is selected on the LINKEDIT control statement (see section 5.6.4.2), a sequential file named SYSUT3 is written by LKED075 and read by LKED077 (see Figure 37, section 7.2.3). This file contains information regarding calls made by each subprogram and is used by LKED077 to produce a cross reference listing.

7.2.1.5.3 Output Files

There are two files output by the linkage editor. One is a file named ØUTPUT which contains a listing of control statements, messages, a storage map, and a cross reference dictionary. Most items scheduled for ØUTPUT are selectable (or suppressed) by options on the LINKEDIT control statement. The second output file contains the executable program. It may be a sequential file on tape or disk, or an indexed file on disk. Its name and status are defined by the ØUTFILE keyword on the LINKEDIT control statement.

APPENDIX C

EXAMPLES OF LINKAGE EDITOR PROCESSING

The following examples have been excerpted from the NASTRAN Programmer's Manual[1]. The SCOPE control cards have been modified to satisfy the requirements of the NSRDC computing system. In these examples, it is assumed that the file containing the call library (LINKLIB) and a file containing the Linkage Editor program (LINKEDT) are contained on separate magnetic tapes.

Example A creates a new user library (NEW) by compiling a source program from input cards. A second user library (OLD) is created by copying previously compiled library decks from the input file. The output of the Linkage Editor is written on a scratch file and executed from that file. This method is most efficient for "compile and go"-type code check runs.

Example B uses a previously compiled user library which is contained on tape. The output of the Linkage Editor is written on tape, but not executed. This type of run should be used when most of the coding errors have been eliminated and the executable link-edited program is saved on tape for subsequent repeated executions.

Example C uses previously compiled binary decks and a tape. Both are used as user libraries. A previously link-edited file (LINKFIL) is modified. The output of the Linkage Editor is written on tape and then executed.

Example D illustrates the link-editing of the program structure shown on page 80.

EXAMPLE A

```
JOB card
CHARGE card
MAP, OFF.
RUN(S,,,,,NEW) or FTN, B = NEW.
REWIND(NEW)
COPYBR(INPUT,OLD,n)
REWIND(OLD)
REQUEST LINKEDT,HI.   (reel #/NORING)
REQUEST LINKLIB,HI.   (reel #/NORING)
LINKEDT.
RETURN (LINKLIB)
RETURN (LINKEDT)
LINKS.ATTACH
7
 8
  9   {FORTRAN or COMPASS source programs}
7
 8
  9   {n object decks}
LINKEDIT OUTFILE=LINKS(R)
LIBRARY NEW,OLD
LINK 0
    {INCLUDE statements}
ENTRY entry point
END
LINK 1
    {INCLUDE, OVERLAY, etc. statements}
ENTRY entry point
END
ENDLINKS
```

```
7
 8
  9  {data for problem program}
7
 8
  9
6
 7
  8
   9
```

EXAMPLE B

```
JOB card
CHARGE card
MAP, OFF.
REQUEST OBJECT,HI.   (reel #/NORING)
REQUEST LINKLIB,HI.   (reel #/NORING)
REQUEST LINKEDT,HI.   (reel #/NORING)
REQUEST LINKFIL,HI.   (reel #/RINGIN)
LINKEDT.
RETURN, OBJECT.
RETURN, LINKLIB.
RETURN, LINKEDT
RETURN, LINKFIL
7
 8
  9
LINKEDIT OUTFILE=LINKFIL(S),LET,XREF,PARAM(7)=2
LIBRARY OBJECT
LINK 0
     {INCLUDE statements for Link 0}
ENTRY  entry point
END
LINK 1
     {INCLUDE, OVERLAY, etc. statements for Link 1}
ENTRY  entry point
END
ENDLINKS
7
 8
  9
6
 7
  8
   9
```

EXAMPLE C

```
JOB card
CHARGE card
MAP, OFF.
COPYBR(INPUT,OBJ,n)
REWIND(OBJ)
REQUEST MASTER,HI. (reel #/NORING)
REQUEST LINKLIB,HI. (reel #/NORING)
REQUEST LINKEDT,HI. (reel #/NORING)
REQUEST LINKFIL,HI. (reel #/RINGIN)
LINKEDT.
RETURN, MASTER.
RETURN, LINKLIB
RETURN, LINKEDT
LINKFIL.
RETURN, LINKFIL
789
      {n object decks}
LINKEDIT  INFILE=LINKFIL(S),OUTFILE=LINKFIL(S),PARAM(6)=90000
LIBRARY  MASTER,OBJ
LINK 2
      {INCLUDE, OVERLAY, etc. statements for Link 2}
ENTRY  entry point
END
ENDLINKS
789
      {data for problem program}
789
6789
```

EXAMPLE D

```
Link 0                    |MAIN
                          |UTIL1
                          |UTIL2
                          |UTIL3
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Link 1                     START
                           MOD1
     ______________________________________________
     | MOD2                                        |
     | MOD3                        B               | MOD4
     | /COM1/                      ____________________________
                                   | MOD5                    MOD7
                                   | MOD6                    /COM3/
                                   | /COM2/
```

```
LIBA                         LIBB
 ____________________        ____________________
|       MAIN         |      |       UTIL1        |
|       UTIL3        |      |       UTIL2        |
|       MOD2         |      |       START        |
|       MOD4         |      |       MOD1         |
|       MOD6         |      |       MOD3         |
|____________________|      |       MOD5         |
                            |       MOD7         |
                            |____________________|
```

The Linkage Editor control commands listed on the opposite page organize LIBA and LIBB into the link-edited structure shown above.

Control Commands

```
LINKEDIT  OUTFILE=LINK(S)
LIBRARY  LIBA,LIBB
LINK 0
INCLUDE  LIBA(MAIN)
INCLUDE  LIBB(UTIL1,UTIL2)
INCLUDE  LIBA(UTIL3)
ENTRY MAIN
END
LINK 1
INCLUDE  LIBB(START,MOD1)
OVERLAY  A
INCLUDE  LIBA(MOD2)
INCLUDE  LIBB(MOD3)
INSERT  COM1
OVERLAY  A
INCLUDE  LIBA(MOD4)
OVERLAY  B
INCLUDE  LIBB(MOD5)
INCLUDE  LIBA(MOD6)
INSERT  COM2
OVERLAY  B
INCLUDE  LIBB(MOD7)
INSERT  COM3
ENTRY START
END
ENDLINKS
```

INITIAL DISTRIBUTION

Copies	
1	DODCI T. Braithwaite
1	ARPA L. Roberts
2	U.S. Army Picatinny Arsenal 1 R. Isakower
1	U.S. Army Frankford Arsenal D. Frederick
1	USAMERDC J. Marburger
4	CNO 1 OP 916 1 OP 916C1, LCDR Poteat 1 OP 916D 1 OP 098TD, L. Aarons
1	CMC
6	CHONR 1 400R, R. Ryan 1 430, R. Lundegard 1 432, L. Bram 1 437, M. Denicoff 1 437, G. Goldstein
1	DNL
6	CHNAVMAT 1 MAT 0141E, R. Jeske 1 MAT 03 1 MAT 03A, CDR Booth 1 MAT 03P2, P. Newton 1 MAT 03P21, S. Atchison
4	USNA 1 D. Rogers 1 A. Adams 1 Dept of Math

Copies	
5	NAVPGSCOL 1 M. Woods 1 D. Williams 1 G. Barksdale 1 C. Comstock
1	NAVWARCOL
1	USNROTC & NAVADMINU, MIT
1	NAVCOSSACT
1	ADPESO
1	CGMCDEC
1	ONR Boston
1	ONR Chicago R. Buchal
1	ONR Pasadena R. Lau
5	NRL 1 5030, S. Wilson 1 5400, B. Wald 1 7810, A. Bligh 1 8050, CDR Tatro
1	COMNAVINT
1	NAVELECSYSCOM
1	NAVSHIPSYSCOM 1 SHIPS 03, RADM Andrews 1 SHIPS 0311, B. Orleans 1 SHIPS 03414, A. Chaikin 1 SHIPS 03423, C. Pohler 1 SHIPS 0719, L. Rosenthal 1 SHIPS 08, Nuclear Power Directorate

Copies

3 NAVAIRSYSCOM
- 1 NAVAIR 5033, R. Saenger
- 1 NAVAIR 5333F4, R. Entner
- 1 NAVAIR 5375A, J. Polgren

1 NAVFACENGCOM

2 NAVORDSYSCOM
- 1 NAVORD 032C, C. McGuigan

2 NAVAIRDEVCEN
- 1 A. Somoroff

1 CIVENGRLAB

10 NELC
- 3 5000, A. Beutel
- 3 5200, M. Lamendola
- 3 5300, J. Dodds

1 NAVUSEACEN

1 NAVWPNSCEN
- L. Diesen

1 NAVCOASTSYSLAB

3 NOL
- 1 R. Edwards
- 1 H. Stevens

6 NWL
- 1 Code K
- 1 Code K-1
- 1 Code KO
- 1 Code KP
- 1 Code KPS

1 NPTLAB NUSC

1 NLONLAB NUSC
- A. Carlson

16 NAVSEC
- 1 SEC 6102C, CDR Anthony
- 1 SEC 6102, CDR Burnett
- 3 SEC 6102C, W. Dietrich
- 1 SEC 6102C, P. Bono
- 1 SEC 6105C1, Y. Park
- 1 SEC 6110.01, R. Leopold
- 1 SEC 6114, R. Johnson
- 1 SEC 6114E, A. Fuller
- 1 SEC 6128, J. O'Brien

Copies

- 1 SEC 6129
- 1 SEC 6133E, E. Straubinger
- 1 SEC 6178D03, L. Biscomb
- 1 SEC 6179A20, J. Singer

1 AFOSR, Code 423

1 Rome Air Development Center

2 WPAFB AFFDL
- 1 J. Johnson

12 DDC

4 NASA Langley Research Center
- 1 R. Butler
- 1 R. Fulton
- 1 J. P. Raney

2 NASA Ames
- 1 P. Pollentz

1 NASA Goddard Space Flight Center
- T. Butler

1 Computer Data Corp

1 Computer Sciences Corp
- D. Roberts

1 Ford Motor Company
- Adv Anal Tech Dept
- P. Anderson

CENTER DISTRIBUTION

Copies	Code
1	1532, E. Baker
2	1725, P. Roth, N. Gifford
1	1735, P. Meyer
1	174, T. Toridis
1	18/1809
1	1802.1
1	1802.2
1	1802.3
1	1802.4
1	1805
1	183
20	1832, R. Martin
1	1833
1	1834
1	1835
1	184
17	1844
	12 J. McKee
	1 M. Golden
	1 M. Hurwitz
	1 B. Kelly
	1 G. Everstine
	1 P. Matula
1	185
1	186
1	188
1	189
2	1891 Central Depository
120	1892.1, S. Good
1	1892.2, D. Sommer
1	1892.3

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Naval Ship Research and Development Center Bethesda, Maryland 20034	UNCLASSIFIED
	2b. GROUP

3. REPORT TITLE

A General Purpose Overlay Loader for CDC 6000-Series Computers; Modification of the NASTRAN Linkage Editor

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (First name, middle initial, last name)

Roger J. Martin

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
April 1973	90	1

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO. ZR0990101	NSRDC 4062
c. 65851N	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d. 1-1844-007	

10. DISTRIBUTION STATEMENT

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

The NASTRAN Linkage Editor is a general purpose linkage editor which provides a means of utilizing available main memory to accommodate large programs which will not fit into the available main memory. As originally designed, the NASTRAN Linkage Editor required RUN FORTRAN compiled input. This report describes a modified and improved version of the Linkage Editor which has been extended to accept either RUN FORTRAN compiled or FORTRAN EXTENDED compiled input.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6801

UNCLASSIFIED

Security Classification

UNCLASSIFIED
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Link Editor						
Loader						
CDC 6000						
Memory Usage						
Overlay Loader						
NASTRAN						
Computer Program						

DD FORM 1 NOV 65 1473 (BACK)
(PAGE 2)

UNCLASSIFIED
Security Classification